Chapter 4

Constrained Particle Swarm Optimisation

The standard Particle Swarm Optimiser is unable to easily optimise functions bound by a set of linear equality or inequality constraints. The objective of this chapter is to present two new algorithms, the Linear Particle Swarm Optimiser (LPSO) and the Converging Linear Particle Swarm Optimiser (CLPSO), designed specifically with constrained optimisation in mind. The properties and convergence of these new algorithms are carefully analysed; a proof for a set of initial conditions on LPSO, a proof of both algorithms' ability to search within the constrained space, and a convergence proof for CLPSO, is given.

4.1 Introduction to constrained optimisation

Optimisation algorithms have the goal of finding the best value of some function. These types of problems are generally composed of three parts: an *objective function* that needs to be optimised (minimised or maximised), a set of solution-defining *variables* on which the objective function depends, and a set of *constraints* that restricts feasible values of these variables. Constraints can be of two types: *equality* constraints specify that a function of the variables must be equal to a constant, while *inequality* constraints specify that a certain function of the variables must be greater than or equal to (or less than or equal to) a constant.

4.1.1 Terminology

Let $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T \in \mathbb{R}^n$ represent the solution-defining variable. This vector $\mathbf{x} \in \mathbb{R}^n$ is called the optimisation variable. The function that needs to be optimised is defined as

$f : \mathbb{R}^n \rightarrow \mathbb{R}$, and is commonly called the objective function or cost function. Let the set of functions $g_i : \mathbb{R}^n \rightarrow \mathbb{R}$ define the inequality constraint functions, giving a set of inequalities $g_i(\mathbf{x}) \leq 0$. Defining $h_i(\mathbf{x}) = 0$ as equality constraints, the functions $h_i : \mathbb{R}^n \rightarrow \mathbb{R}$ are the equality constraint functions. If there are no constraints, the problem is called an unconstrained problem, as was examined in Chapter 3.

Using the above notation, a general optimisation problem can be stated as

$$\begin{aligned} &\text{Minimise} && f(\mathbf{x}) \\ &\text{Subject to} && g_i(\mathbf{x}) \leq 0,\ i = 1 \ldots k \\ & && h_i(\mathbf{x}) = 0,\ i = 1 \ldots m \end{aligned} \tag{4.1}$$

to describe the problem of finding an optimisation variable $\mathbf{x}$ that minimises $f(\mathbf{x})$ over all values of $\mathbf{x}$ that satisfy the conditions $g_i(\mathbf{x}) = 0$, $i = 1 \ldots k$, and $h_i(\mathbf{x}) = 0$, $i = 1 \ldots m$.

The maximum of $f(\mathbf{x})$ can be found by minimising $-f(\mathbf{x})$. In a similar way, an inequality constraint function $g_i(\mathbf{x}) \geq 0$ can be written in the standard form with $-g_i(\mathbf{x}) \leq 0$.

The domain Ω of the constrained optimisation problem is the set of $\mathbf{x}$-values for which the objective and all constraint functions are defined. If $dom(g_i)$ denotes the set of $\mathbf{x}$-values for which $g_i(\mathbf{x}) \leq 0$, and $dom(h_i)$ denotes the set of $\mathbf{x}$-values for which $h_i(\mathbf{x}) = 0$, then Ω is the intersecion of these domains.

$$\Omega = \bigcap_{i=1}^{k} dom(g_i) \cap \bigcap_{i=1}^{m} dom(h_i) \tag{4.2}$$

A point $\mathbf{x} \in \Omega$ is feasible if it satisfies the constraints $g_i(\mathbf{x}) \leq 0$ and $h_i(\mathbf{x}) = 0$.

If Ω is non-empty, there exists at least one feasible point, and the problem is feasible. If no feasible point exists, the problem is infeasible. The domain is the set of all feasible points, called the feasible set. The problem of determining whether the problem is feasible or not is called the feasibility problem. The feasibility problem determines if the inequalities are consistent, and if so, finds a point that satisfies them. It is written as

$$\begin{aligned} &\text{Find} && \mathbf{x} \\ &\text{Subject to} && g_i(\mathbf{x}) \leq 0,\ i = 1 \ldots k \\ & && h_i(\mathbf{x}) = 0,\ i = 1 \ldots m \end{aligned} \tag{4.3}$$

If $\mathbf{x}$ is feasible and $g_i(\mathbf{x}) = 0$, the constraint $g_i(\mathbf{x}) \leq 0$ is *active* at $\mathbf{x}$. If $g_i(\mathbf{x}) < 0$, the constraint $g_i(\mathbf{x}) \leq 0$ is *inactive*. The equality constraint $h_i(\mathbf{x}) = 0$ is active at all feasible points. Redundant constraints are constraints that are implied by other constraints, and deleting them will not change the set of feasible solutions.

The optimal or minimal value f^* of problem (4.1) is defined as

$$f^\star = \inf\{f(\mathbf{x}) \mid g_i(\mathbf{x}) \leq 0,\ i = 1 \ldots k,\ \text{ and }\ h_i(\mathbf{x}) = 0,\ i = 1 \ldots m\} \tag{4.4}$$

The values of $f^\star$ are allowed to take on the extended values $\pm\infty$. If the problem is infeasible, and the standard convention that the infimum of the empty set is $+\infty$ is used, the optimal value $f^\star$ is equal to $+\infty$.

It is also possible that the problem is unbounded from below, such that $f^\star = -\infty$. A problem unbounded from below contains a sequence of feasible points $\{\mathbf{x}_j\}_{j\geq 1}$ with $f(\mathbf{x}_j) \to -\infty$ as $j \to \infty$.

For $\mathbf{x} \in \mathbb{R}^n$ to be an optimal point, it must be feasible and have $f(\mathbf{x}) = f^\star$. The problem may contain more than one feasible $\mathbf{x}$ that minimises $f(\mathbf{x})$, and the set of these optimal points is denoted by

$$\begin{aligned}\mathcal{X} = \{\mathbf{x} \mid\ & g_i(\mathbf{x}) \leq 0,\ i = 1 \ldots k,\ \text{ and} \\ & h_i(\mathbf{x}) = 0,\ i = 1 \ldots m,\ \text{ and }\ f(\mathbf{x}) = f^\star\}\end{aligned} \tag{4.5}$$

If $\mathcal{X}$ is not empty, the optimal value can be found and the problem is solvable. If $\mathcal{X}$ is empty, an optimal value can not be found.

An approximate solution to the problem is very often sufficient if a numeric method is used to find it. This approximate solution must lie within an error margin $\epsilon > 0$ from the true solution. The solution $\mathbf{x}$ with $f(\mathbf{x}) \leq f^\star + \epsilon$ is called ϵ-suboptimal and the set of all ϵ-suboptimal points is called the ϵ-suboptimal set for the problem.

A *local* solution to the optimisation problem is a feasible point $\mathbf{x}$ which will minimise f on the set of nearby feasible solutions within a certain radius from $\mathbf{x}$. A feasible point $\mathbf{x}$ is thus locally optimal if there is an $R > 0$ such that

$$\begin{aligned}f(\mathbf{x}) = \inf\{f(\mathbf{z}) \mid\ & g_i(\mathbf{z}) \leq 0,\ i = 1 \ldots k,\ \text{ and} \\ & h_i(\mathbf{z}) = 0,\ i = 1 \ldots m,\ \text{ and }\ \|\mathbf{z} - \mathbf{x}\| \leq R\}\end{aligned} \tag{4.6}$$

The term 'globally optimal' is used for 'optimal' to distinguish between 'locally optimal' and 'optimal.'

4.1.2 Expressing problems in the standard form

Optimisation problem (4.1) is referred to as a standard form optimisation problem. The convention is chosen that the right-hand side of the inequality and equality constraints are zero. This can always be arranged by subtracting any nonzero right-hand side: for example, the equality constraint $h_i^{(1)}(\mathbf{x}) = h_i^{(2)}(\mathbf{x})$ is written as $h_i(\mathbf{x}) = 0$, where $h_i(\mathbf{x}) = h_i^{(1)}(\mathbf{x}) - h_i^{(2)}(\mathbf{x})$. In a similar way inequalities of the form $g_i(\mathbf{x}) \geq 0$ are expressed as $-g_i(\mathbf{x}) \leq 0$.

4.1.3 Slack variables

Problem (4.1),

$$\begin{aligned} \text{Minimise} \quad & f(\mathbf{x}) \\ \text{Subject to} \quad & g_i(\mathbf{x}) \leq 0, \; i = 1 \ldots k \\ & h_i(\mathbf{x}) = 0, \; i = 1 \ldots m \end{aligned}$$

can be rewritten so that all inequalities involve only a single variable, instead of an entire function $g_i(\mathbf{x})$. These single variables are called *slack variables* and replaces each inequality constraint with an equality constraint, and a non-negativity constraint. There is one slack variable s_i associated with each original inequality constraint $g_i(\mathbf{x}) \leq 0$. Optimisation problem (4.1) is rewritten as

$$\begin{aligned} \text{Minimise} \quad & f(\mathbf{x}) \\ \text{Subject to} \quad & g_i(\mathbf{x}) + s_i = 0, \; i = 1 \ldots k \\ & h_i(\mathbf{x}) = 0, \quad i = 1 \ldots m \\ & s_i \geq 0, \quad i = 1 \ldots k \end{aligned} \tag{4.7}$$

where the variables are $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{s} \in \mathbb{R}^k$. This problem has $n + k$ variables, k inequality constraints (the non-negativity constraints on s_i), and $k + m$ equality constraints.

The problem is equivalent to the original standard form problem. If $(\mathbf{x}, \mathbf{s})$ is feasible for the above problem, then $\mathbf{x}$ is feasible for the original problem, since $s_i = -g_i(\mathbf{x}) \geq 0$. Conversely, if $\mathbf{x}$ is feasible for the original problem, then $(\mathbf{x}, \mathbf{s})$ is feasible for the above problem, where $s_i = -g_i(\mathbf{x})$. Similarly, $\mathbf{x}$ is optimal for the original problem if and only if $(\mathbf{x}, \mathbf{s})$ is optimal for the above problem, where $s_i = -g_i(\mathbf{x})$.

4.1.4 Convex optimisation

A convex optimisation problem is one of the form

$$\begin{aligned} \text{Minimise} \quad & f(\mathbf{x}) \\ \text{Subject to} \quad & g_i(\mathbf{x}) \leq 0, \; i = 1 \ldots k \\ & A\mathbf{x} = \mathbf{b}, \; A \in \mathbb{R}^{m \times n} \text{ and } \mathbf{b} \in \mathbb{R}^m \end{aligned} \tag{4.8}$$

where both f and $g_1, \ldots, g_k$ are convex functions. The convex problem has three additional requirements compared to the general standard form problem (4.1): the objective is convex, the inequality constraint functions are convex, and the equality constraint functions $h_i(\mathbf{x}) = \mathbf{a}_i^T \mathbf{x} - b_i$ are affine.

The additional requirements give rise to an important property: the feasible set of a convex optimisation problem is convex, since it is the domain of the problem

$$\Omega = \bigcap_{i=1}^{k} dom(g_i) \cap \{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}\} \tag{4.9}$$

which is a convex set, with k (convex) sublevel sets $\{\mathbf{x} \mid g_i(\mathbf{x}) \leq 0\}$ and m hyperplanes $\{\mathbf{x} \mid \mathbf{a}_i^T\mathbf{x} = b_i\}$.

An important property of convex optimisation problems is that any locally optimal point is also globally optimal. To see this, suppose that (feasible) $\mathbf{x}$ is locally optimal for a convex optimisation problem, and

$$\begin{aligned} f(\mathbf{x}) = \inf\{f(\mathbf{z}) \mid g_i(\mathbf{z}) \leq 0,\ i = 1 \ldots k,\ \text{and} \\ \mathbf{a}_i^T\mathbf{x} = b_i,\ i = 1 \ldots m,\ \text{and}\ \|\mathbf{z} - \mathbf{x}\| \leq R\} \end{aligned} \tag{4.10}$$

for some $R > 0$. Now suppose that $\mathbf{x}$ is not globally optimal, such that there is a feasible $\mathbf{y}$ with $f(\mathbf{y}) < f(\mathbf{x})$. As a result $\|\mathbf{y} - \mathbf{x}\| > R$, since otherwise $f(\mathbf{x}) \leq f(\mathbf{y})$. Consider the point $\mathbf{z}$ given by

$$\mathbf{z} = (1 - \alpha)\mathbf{x} + \alpha\mathbf{y}, \quad \alpha = \frac{R}{2\|\mathbf{y} - \mathbf{x}\|} \tag{4.11}$$

It follows that $\|\mathbf{z} - \mathbf{x}\| = \frac{R}{2} < R$, and by convexity of the feasible set, $\mathbf{z}$ is feasible. By convexity of f it follows that

$$f(\mathbf{z}) \leq (1 - \alpha)f(\mathbf{x}) + \alpha f(\mathbf{y}) < f(\mathbf{x}) \tag{4.12}$$

which contradicts (4.10). Hence there exists no feasible $\mathbf{y}$ with $f(\mathbf{y}) < f(\mathbf{x})$, and it follows that $\mathbf{x}$ is globally optimal.

4.1.5 Duality

Consider the standard optimisation problem (4.1), with a non-empty domain Ω, also called a 'primal problem.' The constraints in (4.1) can be introduced to the objective function by augmenting it with a weighted sum of the constraint functions. Let the vector $\boldsymbol{\mu} \in \mathbb{R}^k$ be associated with the set of k inequality constraints, and $\boldsymbol{\lambda} \in \mathbb{R}^m$ be associated with the set of m equality constraints. These vectors are called the Lagrange multiplier vectors, and define the Lagrangian $L : \mathbb{R}^n \times \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}$ associated with (4.1) as

$$L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = f(\mathbf{x}) + \sum_{i=1}^{k} \mu_i g_i(\mathbf{x}) + \sum_{i=1}^{m} \lambda_i h_i(\mathbf{x}) \tag{4.13}$$

The dual problem associated with the primal problem is written as

$$\begin{aligned} &\text{Maximise } L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \text{ with respect to } \boldsymbol{\mu} \text{ and } \boldsymbol{\lambda} \\ &\text{Subject to } \quad \boldsymbol{\mu} \geq 0 \end{aligned} \tag{4.14}$$

If the problem (4.1) is convex, then the solution of the primal problem is the vector $\mathbf{x}^\star$ of the saddle point $(\mathbf{x}^\star, \boldsymbol{\mu}^\star, \boldsymbol{\lambda}^\star)$ of (4.13) such that

$$L(\mathbf{x}^\star, \boldsymbol{\mu}, \boldsymbol{\lambda}) \leq L(\mathbf{x}^\star, \boldsymbol{\mu}^\star, \boldsymbol{\lambda}^\star) \leq L(\mathbf{x}, \boldsymbol{\mu}^\star, \boldsymbol{\lambda}^\star) \tag{4.15}$$

The vector $\mathbf{x}^\star$ that solves the primal problem, as well as the two Lagrange multiplier vectors $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$, can be found by solving the min-max problem

$$\min_{\mathbf{x}} \max_{\boldsymbol{\mu}, \boldsymbol{\lambda}} L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \tag{4.16}$$

4.1.6 Equality-constrained optimisation

The final optimisation problem introduced, is the problem of minimising f under a set of linear equality constraints. This problem is written as

$$\begin{aligned} &\text{Minimise } \quad f(\mathbf{x}) \\ &\text{Subject to } A\mathbf{x} = \mathbf{b}, \; A \in \mathbb{R}^{m \times n} \text{ and } \mathbf{b} \in \mathbb{R}^m \end{aligned} \tag{4.17}$$

and will form the basis of PSO developments to follow. In the following sections, a PSO algorithm is developed to successfully handle the above equality constrained optimisation problem. The method is extended to handle inequality constraints as well.

4.2 Linear Particle Swarm Optimisation

The Particle Swarm Optimisation (PSO) algorithm discussed in Chapter 3 is an algorithm suited for unconstrained optimisation. This section introduces a new PSO algorithm, the Linear Particle Swarm Optimiser, that is specifically developed with linear constraints in mind. The update equations for a particle's velocity and position, with inertia weight w, is repeated here:

$$v_{ij}^{(t+1)} = w v_{ij}^{(t)} + c_1 r_1^{(t)} [z_{ij}^{(t)} - p_{ij}^{(t)}] + c_2 r_2^{(t)} [\hat{z}_j^{(t)} - p_{ij}^{(t)}] \tag{4.18}$$

$$p_{ij}^{(t+1)} = v_{ij}^{(t+1)} + p_{ij}^{(t)} \tag{4.19}$$

Traditionally, the above velocity and position update steps are specified separately for each dimension of a particle, as is done in [26, 28, 49, 55]. If the random numbers $r_1^{(t)}$ and $r_2^{(t)}$

are rather kept constant for all vector dimensions, the velocity updates are calculated as a linear combination of position and velocity vectors.

$$\mathbf{v}_i^{(t+1)} = w\mathbf{v}_i^{(t)} + c_1 r_1^{(t)}[\mathbf{z}_i^{(t)} - \mathbf{p}_i^{(t)}] + c_2 r_2^{(t)}[\widehat{\mathbf{z}}^{(t)} - \mathbf{p}_i^{(t)}] \tag{4.20}$$

$$\mathbf{p}_i^{(t+1)} = \mathbf{v}_i^{(t+1)} + \mathbf{p}_i^{(t)} \tag{4.21}$$

The above approach has the advantage that the flight of particles is defined by standard linear operations on vectors. The guaranteed movement of particles through subspaces and subsets becomes possible, as stated in Theorem 4.1 (to follow). The PSO algorithm using update equations (4.20, 4.21) is referred to as a "Linear Particle Swarm Optimiser" (LPSO), due to the way the update equations are formulated. The LPSO algorithm, used to minimise a function

$$f : \mathbb{R}^n \to \mathbb{R} \tag{4.22}$$

is presented below:

Algorithm 4.1 - Linear Particle Swarm Optimisation (LPSO)

1. Set the iteration number t to zero, and randomly initialise the swarm S of n-dimensional particles $\mathbf{p}_i^{(0)}$ to a value in the initial domain of the swarm. Initialise all velocity vectors $\mathbf{v}_i = \mathbf{0}$.

2. Evaluate the performance $f(\mathbf{p}_i^{(t)})$ of each particle.

3. Compare the personal best of each particle to its current performance, and set $\mathbf{z}_i^{(t)}$ to the better performance:

$$\mathbf{z}_i^{(t)} = \begin{cases} \mathbf{z}_i^{(t-1)} & \text{if } f(\mathbf{p}_i^{(t)}) \geq f(\mathbf{z}_i^{(t-1)}) \\ \mathbf{p}_i^{(t)} & \text{if } f(\mathbf{p}_i^{(t)}) < f(\mathbf{z}_i^{(t-1)}) \end{cases} \tag{4.23}$$

4. Set the global best $\widehat{\mathbf{z}}^{(t)}$ to the position of the best performance in the swarm:

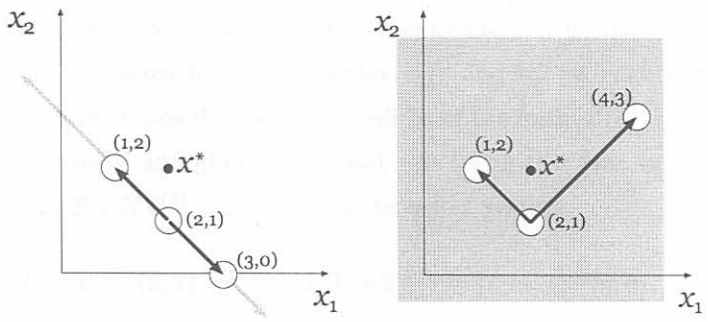
$$\begin{aligned} \widehat{\mathbf{z}}^{(t)} &\in \{\mathbf{z}_1^{(t)}, \mathbf{z}_2^{(t)}, \ldots, \mathbf{z}_s^{(t)}\} \mid f(\widehat{\mathbf{z}}^{(t)}) \\ &= \min\{f(\mathbf{z}_1^{(t)}), f(\mathbf{z}_2^{(t)}), \ldots, f(\mathbf{z}_s^{(t)})\} \end{aligned} \tag{4.24}$$

5. Change the velocity vector for each particle according to equation (4.20).

6. Move each particle to its new position, according to equation (4.21).

7. Let $t := t + 1$.

8. Go to step 2, and repeat until convergence or $t = t_{max}$.



(a) S^* spanning $\mathbb{R}^1$. (b) S^* spanning $\mathbb{R}^2$.

FIGURE 4.1: Comparing the possible search spaces resulting from different initial swarms in LPSO, with $\mathbf{v}_i^{(0)} = \mathbf{0}$.

The algorithm has converged if the difference between the best solution found over a specified number of iterations remains within a certain bound. The algorithm iterates until either one of two conditions is met: the algorithm has converged, or the maximum number of iterations t_{max} have been reached. In essence the convergence and stopping conditions are therefore the same as for the standard PSO.

4.2.1 Criteria on the initial swarm

If a PSO is considered in the traditional sense, with random numbers $r_1^{(t)}$ and $r_2^{(t)}$ generated for each dimension in a particle's update equations (4.18, 4.19), any point in the search space can possibly be reached with a swarm of arbitrary size. It is even possible to reach any point in the search space with a swarm of size two [28].

This generalisation does not work for the LPSO, where the update equations (4.20) and (4.21) are in fact linear combinations of position and velocity vectors. The initial swarm will thus influence which positions can and cannot be found.

In fact, if all velocities are initialised to zero (like in the LPSO algorithm above), only positions in the span of the set of vectors created by subtracting the initial global best $\hat{\mathbf{z}}^{(0)}$ from each initial position vector, will be found. A similar idea is true if the initial velocities are non-zero, where the initial velocity vectors are added to the previous set of vectors (created by subtracting the global best $\hat{\mathbf{z}}^{(0)}$ from each initial position vector) to span the set of possible solutions found.

Consider the example illustrated in Figures 4.1(a) and 4.1(b), and say $f(\mathbf{x})$ is minimized at a point (or vector) $\mathbf{x}^*$. If the LPSO algorithm is able to find $\mathbf{x}^*$, vector $\mathbf{x}^*$ should be decomposable into a linear combination of the initial velocity vectors.

It is easy to see from Figure 4.1(a) that a swarm with initial population $\{(1,2),\ (2,1),\ (3,0)\}$ will never be able to reach $\mathbf{x}^\star = (2,2)$. This is due to the way the particles are moved with velocities which are initialised to the zero vector. $\mathbf{v}_1^{(t)}$ and $\mathbf{v}_2^{(t)}$ will cause the particles to fly on a straight line, since all possible velocities will be of the form $\alpha[(1,2)-(2,1)] = \alpha(-1,1)$, with $\alpha \in \mathbb{R}$. All personal and global bests will also lie on this line, and thus searches will be in $\mathbb{R}^1$ and not in $\mathbb{R}^2$. If $\widehat{\mathbf{z}}^{(0)} = (2,1)$, then the set of vectors

$$\{(1,2)-\widehat{\mathbf{z}}^{(0)},\ (2,1)-\widehat{\mathbf{z}}^{(0)},\ (3,0)-\widehat{\mathbf{z}}^{(0)}\} = \{(-1,1),\ (0,0),\ (1,-1)\}$$

as shown in Figure 4.1(a), will only span $\mathbb{R}^1$. The optimal value $\mathbf{x}^\star$ at $(2,2)$ can not be reached. In comparison, Figure 4.1(b) shows that the set of vectors

$$\{\mathbf{p}-\widehat{\mathbf{z}}^{(0)} \mid \mathbf{p} \in S^{(0)}\} = \{(-1,1),\ (0,0),\ (2,2)\}$$

from the initial swarm $S^{(0)} = \{(1,2),\ (2,1),\ (4,3)\}$ will span $\mathbb{R}^2$, and $\mathbf{x}^\star$ at $(2,2)$ can possibly be reached.

This leads us to a first important theorem, which makes the following assumptions from Algorithm 4.1 (LPSO):

1. $\mathbf{v}_i^{(0)} = \mathbf{0}$
2. $\mathbf{z}_i^{(0)} = \mathbf{p}_i^{(0)}$

Theorem 4.1

If f needs to be optimised in $\mathbb{R}^n$, a swarm of s particles $S^{(0)} = \{\mathbf{p}_1^{(0)},\ \mathbf{p}_2^{(0)},\ \ldots, \mathbf{p}_s^{(0)}\}$ will be able to find the optimal value $\mathbf{x}^\star$ if and only if there exists a subset $S^\star \subseteq S^{(0)} - \widehat{\mathbf{z}}^{(0)} = \{\mathbf{p}-\widehat{\mathbf{z}}^{(0)} \mid \mathbf{p} \in S^{(0)}\}$ that forms a basis for $\mathbb{R}^n$.

Theorem 4.2 is used to prove Theorem 4.1, and thus the proof of Theorem 4.1 is 'postponed' to follow the proof of Theorem 4.2, and follows in Section 4.3.2.

Since one of the particles will be the global best particle with $\mathbf{p}_i^{(0)} - \widehat{\mathbf{z}}^{(0)} = \mathbf{0}$, the set of vectors $S^{(0)} - \widehat{\mathbf{z}}^{(0)}$ will contain the zero vector, and so $S^{(0)}$ needs to contain a minimum of $n+1$ vectors for $S^{(0)} - \widehat{\mathbf{z}}^{(0)}$ to span $\mathbb{R}^n$, namely

$$\inf |S^{(0)}| = n+1 \tag{4.25}$$

To explore the case when initial velocities are non-zero, consider the LPSO update equations (4.20) and (4.21). Assuming that the initial personal best $\mathbf{z}_i^{(0)}$ is set to $\mathbf{p}_i^{(0)}$, two vectors play a role in particle i's update equations: the initial velocity vector $\mathbf{v}_i^{(0)}$ and the difference between the initial global best $\widehat{\mathbf{z}}^{(0)}$ and the initial position $\mathbf{p}_i^{(0)}$. It follows that the set of vectors

$$\{\mathbf{v}_1^{(0)}, \widehat{\mathbf{z}}^{(0)} - \mathbf{p}_1^{(0)}, \mathbf{v}_2^{(0)}, \widehat{\mathbf{z}}^{(0)} - \mathbf{p}_2^{(0)}, \ldots, \mathbf{v}_s^{(0)}, \widehat{\mathbf{z}}^{(0)} - \mathbf{p}_s^{(0)}\}$$

must span $\mathbb{R}^n$, and the minimum swarm size for LPSO of $S^{(0)}$ will be $\lceil \frac{n}{2} \rceil + 1$.

4.3 Equality-constrained optimisation

The LPSO algorithm lends itself perfectly to solving equality-constrained optimisation problems, as was discussed in Section 4.1.6. This section summarises current methods from the Evolutionary Computing and PSO fields, and discusses and proves the usefulness of LPSO to equality-constrained optimisation.

4.3.1 Current methods

Many methods for constraint handling have been proposed in the Evolutionary Computation field [33]. These can be broadly classified into *penalty*, *repair* and *constraint-preserving* methods.

Penalty methods add a penalty to the objective function to decrease the quality of infeasible solutions [21, 23, 33]. While penalty methods are very popular, they do not guarantee a solution where no constraints are violated, since the search space still includes infeasible solutions, and success depends on the penalty method.

Repair methods apply operators to move an infeasible solution closer to the feasible space of solutions [31, 62]. Operators designed to 'correct' infeasible solutions are usually computationally intensive. Not all constraints can be easily implemented to be corrected by these operators, which must be tailored to the particular problem [16].

Constraint-preserving methods (feasible solutions methods) reduce the search space by ensuring that all candidate solutions at all times satisfy the constraints [33]. Solutions are initialised within the feasible domain, and transformations of candidate solutions are such that the resulting solutions still lie within the feasible domain.

Hamida and Schoenauer introduced a hybrid approach for Evolutionary Algorithms to handle equality constraints [23]. In this approach, equalities $h_j(\mathbf{x}) = 0$ are written as double inequalities $-\varepsilon^{(t)} \leq h_j(\mathbf{x}) \leq \varepsilon^{(t)}$. The idea is to start, for each equality, with a large feasible domain, and so tolerate high violation degrees. This domain is then gradually reduced along evolution, in order to bring it as close as possible to a null measure feasible domain, as illustrated in Figure 4.2. The value of ε is progressively reduced with the aim of reaching $0 \leq h_j(\mathbf{x}) \leq 0$.

Feasible solutions methods, on the other hand, are based on transforming feasible individuals into other feasible individuals. In the Evolutionary Algorithm sense, it is done by

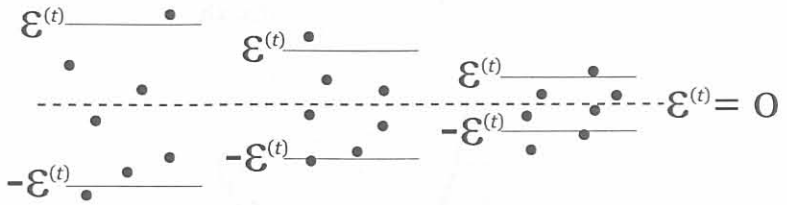


FIGURE 4.2: Progressive reduction of the feasible domain.

operators that are closed on the feasible part of the search space. These methods assume linear constraints only and a feasible starting point, or a feasible initial population [33].

Michalewicz and Janikow developed a genetic algorithm called *Genocop*, named after "GEnetic algorithm for Numerical Optimisation for COnstrained Problems" [32]. The approach, focusing on linear constrains, firstly eliminates the equalities in the set of constraints, and secondly uses carefully designed 'genetic' operators that guarantee to keep all 'chromosomes' of the genetic algorithm within the constrained space.

Shi and Krohling developed a method using two co-evolving PSOs, and duality from Section 4.1.5, to solve a constrained optimisation problem [50]. The min-max problem (4.16) is solved by evolving two simultaneous PSOs. The first PSO freezes the Lagrange multipliers $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$, and minimises the Lagrangian $L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ over $\mathbf{x}$. The second PSO freezes the variable vector $\mathbf{x}$, and maximises $L(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ over the Lagrange multipliers $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$. However, if the optimisation problem is non-convex, the solution of the primal and dual problems do not coincide. In this case a penalty, determined by the inequality and equality constraint functions, is added to the Lagrangian.

The LPSO falls in the constraint-preserving class of constraint handling algorithms. Linear constraints are assumed, and if the initial swarm contains only feasible starting points, transitions to new solutions through velocity updates ensure feasible solutions to be generated.

4.3.2 PSO for equality-constrained optimisation

Let the objective be to find the minimum of some function $f(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^n$, subject to a set of linear constraints,

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n &= b_2 \\ &\ldots \\ a_{m1}x_1 + a_{m2}x_2 + \ldots + a_{mn}x_n &= b_m \end{aligned}$$

or

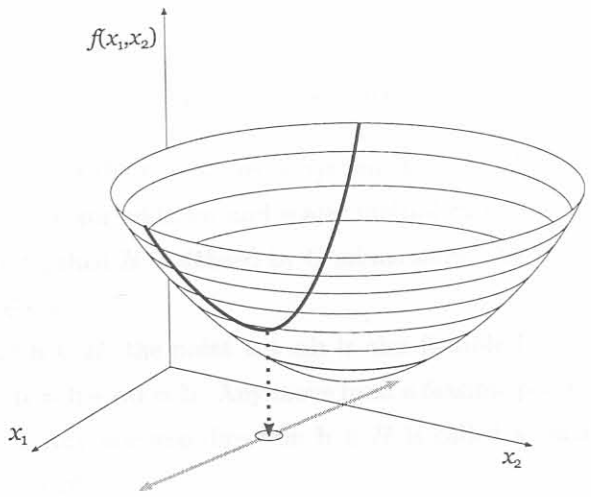


FIGURE 4.3: Minimising f under a linear equality constraint.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

It is assumed that the problem is feasible, or the solution set for the linear constraints is non-empty. Then, in simple terms, the problem is defined as

$$\begin{aligned} &\text{Minimise} \quad f(\mathbf{x}), \quad \mathbf{x} \in \mathbb{R}^n \\ &\text{Subject to } A\mathbf{x} = \mathbf{b},\ A \in \mathbb{R}^{m \times n} \text{ and } \mathbf{b} \in \mathbb{R}^m \end{aligned} \tag{4.26}$$

It can be said that f needs to be optimised in the hyperplane C, the set of particular solutions of the linear system $A\mathbf{x} = \mathbf{b}$. That is,

$$C = \{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}\}$$

defines the set of feasible solutions to (4.26), and each point in C will be a feasible point. Figure 4.3 illustrates a one-dimensional hyperplane (or line) C that constrains two-dimensional solutions $\mathbf{x} = (x_1, x_2)$.

The approach presented below flies the swarm through the set of feasible solutions, in this case hyperplane C.

Feasible directions

Given a feasible point $\mathbf{x}$ (a particle's position, for instance), it will be necessary to fly from $\mathbf{x}$ to other feasible points. This can be done with feasible directions. Let

$$H = \{\mathbf{x} \mid A\mathbf{x} = \mathbf{0}\}$$

define the set of solutions of the homogeneous system $A\mathbf{x} = \mathbf{0}$. H is a subspace of $\mathbb{R}^n$, and since H is closed under vector addition and scalar multiplication, it is also a vector space. If $\mathbf{c}_0$ is any element of C, then H is defined by C minus some offset $\mathbf{c}_0$, or the set of vectors $C - \mathbf{c}_0 = \{\mathbf{c} - \mathbf{c}_0 \mid \mathbf{c} \in C\}$.

If $\mathbf{x}$ is feasible and $\mathbf{h} \in H$, the point $\mathbf{x} + \alpha\mathbf{h}$ is also feasible for every value of α, since $A(\mathbf{x}+\alpha\mathbf{h}) = A\mathbf{x}+\alpha A\mathbf{h} = \mathbf{b}+\alpha\mathbf{0} = \mathbf{b}$. Any move from a feasible point along $\mathbf{h}$ will produce another feasible point. Any nonzero direction $\mathbf{h} \in H$ is called a *feasible direction* for the constraints $A\mathbf{x} = \mathbf{b}$ in (4.26).

If the initial swarm is feasible, and the particles fly with only feasible directions as their velocity vectors, then the swarm will stay within the search space. This is summarized in Theorem 4.2, which can be proved by a simple inductive argument:

Theorem 4.2

If all initial velocity vectors $\mathbf{v}_i^{(0)}$ *are solutions to the homogeneous system* $A\mathbf{x} = \mathbf{0}$, *and all initial particles* $\mathbf{p}_i^{(0)}$ *lie in the hyperplane defined by* $A\mathbf{x} = \mathbf{b}$, *then for any time* t

$$\begin{aligned} &I) && A\mathbf{v}_i^{(t)} = \mathbf{0} \\ &II) && A\mathbf{p}_i^{(t)} = \mathbf{b} \\ &III) && A\mathbf{z}_i^{(t)} = \mathbf{b} \\ &IV) && A\widehat{\mathbf{z}}^{(t)} = \mathbf{b} \end{aligned}$$

i.e. the swarm will fly through the hyperplane defined by the constraints.

Proof

Without losing generality, subscript i, denoting a specific particle in the swarm, is dropped.

Basis step:

I) $\mathbf{v}^{(0)} = \mathbf{0}$ (by initialisation) is the trivial solution to $A\mathbf{x} = \mathbf{0}$

II) $\mathbf{p}^{(0)}$ is initialised on the hyperplane $A\mathbf{x} = \mathbf{b}$

III) $\mathbf{z}^{(0)} = \mathbf{p}^{(0)}$

$\Rightarrow A\mathbf{z}^{(0)} = \mathbf{b}$

IV) $\widehat{\mathbf{z}}^{(0)} \in \{\mathbf{z}_1^{(0)}, \mathbf{z}_2^{(0)}, \ldots, \mathbf{z}_s^{(0)}\}$

$\mid f(\widehat{\mathbf{z}}^{(0)}) = \min\{ f(\mathbf{z}_1^{(0)}), f(\mathbf{z}_2^{(0)}), \ldots, f(\mathbf{z}_s^{(0)})\}$

$\Rightarrow A\widehat{\mathbf{z}}^{(0)} = \mathbf{b}$

Inductive step:

Suppose $A\mathbf{v}^{(k)} = \mathbf{0}$, $A\mathbf{p}^{(k)} = \mathbf{b}$, $A\mathbf{z}^{(k)} = \mathbf{b}$ and $A\widehat{\mathbf{z}}^{(k)} = \mathbf{b}$. Then

I) $A\mathbf{v}^{(k+1)}$
$= A\big(w\mathbf{v}^{(k)} + c_1 r_1^{(k)}[\mathbf{z}^{(k)} - \mathbf{p}^{(k)}] + c_2 r_2^{(k)}[\widehat{\mathbf{z}}^{(k)} - \mathbf{p}^{(k)}]\big)$
$= wA\mathbf{v}^{(k)} + c_1 r_1^{(k)}(A\mathbf{z}^{(k)} - A\mathbf{p}^{(k)}) + c_2 r_2^{(k)}(A\widehat{\mathbf{z}}^{(k)} - A\mathbf{p}^{(k)})$
$= wA\mathbf{v}^{(k)} + c_1 r_1^{(k)}(\mathbf{b} - \mathbf{b}) + c_2 r_2^{(k)}(\mathbf{b} - \mathbf{b})$
$= w\mathbf{0} + c_1 r_1^{(k)}\mathbf{0} + c_2 r_2^{(k)}\mathbf{0}$
$= \mathbf{0}$

II) $A\mathbf{p}^{(k+1)} = A(\mathbf{v}^{(k+1)} + \mathbf{p}^{(k)})$
$= A\mathbf{v}^{(k+1)} + A\mathbf{p}^{(k)}$
$= \mathbf{0} + \mathbf{b}$
$= \mathbf{b}$

III) $A\mathbf{z}^{(k+1)} = \begin{cases} A\mathbf{z}^{(k)} & \text{if } f(\mathbf{p}^{(k+1)}) \geq f(\mathbf{z}^{(k)}) \\ A\mathbf{p}^{(k+1)} & \text{if } f(\mathbf{p}^{(k+1)}) < f(\mathbf{z}^{(k)}) \end{cases}$
$= \mathbf{b}$

IV) $\widehat{\mathbf{z}}^{(k+1)} \in \{\mathbf{z}_1^{(k+1)}, \mathbf{z}_2^{(k+1)}, \ldots, \mathbf{z}_s^{(k+1)}\} \mid f(\widehat{\mathbf{z}}^{(k+1)})$
$= \min\{f(\mathbf{z}_1^{(k+1)}), f(\mathbf{z}_2^{(k+1)}), \ldots, f(\mathbf{z}_s^{(k+1)})\}$
$\Rightarrow A\widehat{\mathbf{z}}^{(k+1)} = \mathbf{b}$

□

This shows that the swarm will fly through the solution hyperplane C defined by the set of feasible solutions.

Theorem 4.1 can now be proved using the result from Theorem 4.2. Assuming $\mathbf{v}_i^{(0)} = \mathbf{0}$ and $\mathbf{z}_i^{(0)} = \mathbf{p}_i^{(0)}$ from the LPSO algorithm (Algorithm 4.1), Theorem 4.1 stated that:

If f needs to be optimised in $\mathbb{R}^n$, a swarm of s particles $S^{(0)} = \{\mathbf{p}_1^{(0)}, \mathbf{p}_2^{(0)}, \ldots, \mathbf{p}_s^{(0)}\}$ will be able to find the optimal value $\mathbf{x}^\star$ if and only if there exists a subset $S^\star \subseteq S^{(0)} - \widehat{\mathbf{z}}^{(0)} = \{\mathbf{p} - \widehat{\mathbf{z}}^{(0)} \mid \mathbf{p} \in S^{(0)}\}$ that forms a basis for $\mathbb{R}^n$.

Proof of Theorem 4.1

Assume $\mathbf{x}^\star$ can be reached. Then particles must be able to traverse the entire $\mathbb{R}^n$ to reach it. If particles are searching in $\mathbb{R}^n$, they are searching in unconstrained space. Let $A \in \mathbb{R}^{n \times n}$. An unconstrained space is equivalent to a space constrained with $A\mathbf{x} = \mathbf{b} = \mathbf{0}$ only if $rank(A) = 0$ (implying *any* $\mathbf{x}$ is a feasible solution, which is exactly $\mathbb{R}^n$).

We will show that if $rank(A) = 0$, then there must exist a subset of vectors from $\{\mathbf{p}_1^{(0)} - \widehat{\mathbf{z}}^{(0)}, \ldots, \mathbf{p}_s^{(0)} - \widehat{\mathbf{z}}^{(0)}\}$ that forms a basis for $\mathbb{R}^n$.

Let the rank of A be zero. For each particle, $A\mathbf{p}_i^{(0)} = \mathbf{0}$ and $A\widehat{\mathbf{z}}^{(0)} = \mathbf{0}$, and thus

$$A(\mathbf{p}_i^{(0)} - \widehat{\mathbf{z}}^{(0)}) = \mathbf{0}, \qquad i = 1 \ldots s$$

Let $\mathbf{u}_i = \mathbf{p}_i^{(0)} - \widehat{\mathbf{z}}^{(0)}$, such that

$$A\mathbf{u}_i = \mathbf{0}, \qquad i = 1 \ldots s$$

The kernel of A is defined as the linear space

$$ker(A) = \{\mathbf{x} \mid A\mathbf{x} = \mathbf{0}\}$$

with $ker(A) \subseteq \mathbb{R}^n$. Since $A\mathbf{u}_i = \mathbf{0}$, we have $\mathbf{u}_i \in ker(A)$. From the definition of $ker(A)$, if $\mathbf{u}_1 \ldots \mathbf{u}_s$ span $\mathbb{R}^n$, then

$$dim(ker(A)) = n$$

and if $\mathbf{u}_1 \ldots \mathbf{u}_s$ do not span $\mathbb{R}^n$, then

$$dim(ker(A)) \leq n$$

The dimension of $ker(A)$ is therefore strictly equal to n, only if $\mathbf{u}_1 \ldots \mathbf{u}_s$ span $\mathbb{R}^n$. Since

$$dim(ker(A)) + rank(A) = dim(A)$$

the rank of A is stricltly equal to zero only if $\mathbf{u}_1 \ldots \mathbf{u}_s$ span $\mathbb{R}^n$. It follows that there are n linearly independent vectors in $\mathbf{u}_1 \ldots \mathbf{u}_s$ that form a basis for $\mathbb{R}^n$.

To prove the converse, assume that a subset of $\mathbf{u}_1 \ldots \mathbf{u}_s$ forms a basis for $\mathbb{R}^n$. Then we can define a matrix $A \in \mathbb{R}^{n \times n}$ such that $A\mathbf{u}_i = \mathbf{0}$ for each $\mathbf{u}_i$, and again $\mathbf{u}_i \in ker(A)$. Thus we have $dim(ker(A)) = n$, implying $rank(A) = 0$. From Theorem 4.2, the swarm of particles will 'fly' through the unconstrained search space $\mathbb{R}^n$ (since $rank(A) = 0$, and therefore all initial particles $\mathbf{p}_i^{(0)}$ meet $A\mathbf{x} = \mathbf{b} = \mathbf{0}$). Any point in $\mathbb{R}^n$, including $\mathbf{x}^\star$, can therefore be reached. □

Change of PSO for constrained optimisation

It is clear from the above that, if the swarm is initialised to a set of feasible solutions, all solutions found will be feasible. However, this does not mean that the optimum solution can be found.

Theorem 4.1 provides a condition on the initial swarm that guarantees that any point inside the search space can be found. This search space was $\mathbb{R}^n$. With the given constraints, the search space will be some hyperplane inside $\mathbb{R}^n$. The initial swarm can be chosen such that any point in this hyperplane can be found.

By definition, any direction $\mathbf{h}$ satisfying $A\mathbf{h} = \mathbf{0}$ lies in the null space of A. If the rank of A is r, let

$$S^\star = \{\mathbf{p}_1^{(0)} - \hat{\mathbf{z}}^{(0)},\ \mathbf{p}_2^{(0)} - \hat{\mathbf{z}}^{(0)},\ \ldots,\ \mathbf{p}_{n-r}^{(0)} - \hat{\mathbf{z}}^{(0)}\}$$

denote a generic set of $n - r$ linearly independent vectors, such that $A(\mathbf{p}_i^{(0)} - \widehat{\mathbf{z}}^{(0)}) = A\mathbf{p}_i^{(0)} - A\widehat{\mathbf{z}}^{(0)} = \mathbf{b} - \mathbf{b} = \mathbf{0}$. This implies that $S^\star$ forms a basis for the $n - r$ dimensional null space of A. $S^\star$ provides a convenient way to represent all feasible points. Given any point $\mathbf{p}^{(0)}$ such that $A\mathbf{p}^{(0)} = \mathbf{b}$, every feasible point can be written as $\mathbf{p}^{(0)}$ plus some linear combination of $S^\star$.

For constrained optimisation, f is optimised in an $n - r$ dimensional hyperplane inside $\mathbb{R}^n$, with $r = rank(A)$. Thus a swarm of s particles $S^{(0)} = \{\mathbf{p}_1^{(0)}, \mathbf{p}_2^{(0)}, \ldots, \mathbf{p}_s^{(0)}\}$ will be able to find the optimal value if and only if there exists a subset $S^\star \subseteq S^{(0)} - \widehat{\mathbf{z}}^{(0)} = \{\mathbf{p} - \widehat{\mathbf{z}}^{(0)} \mid \mathbf{p} \in S^{(0)}\}$ that forms a basis for $\mathbb{R}^{n-r}$. In this case the minimum swarm size will be

$$\inf |S^{(0)}| = n - r + 1 \tag{4.27}$$

If the whole swarm is thus initialised to lie within the hyperplane $A\mathbf{x} = \mathbf{b}$, and $S^\star \subseteq S^{(0)} - \widehat{\mathbf{z}}^{(0)}$ defines a basis for $\mathbb{R}^{n-r}$, then f can be optimised in the standard way. It is due to this property that Linear Particle Swarm Optimisation is ideally suited to solving these kinds of optimisation problems.

Initialising particles within the search plane

The next task is to find a way to initialise such a swarm with s particles. Most importantly, all particles should lie within the search plane. This can be done by reducing the augmented matrix $(A|\mathbf{b})$ to row-echelon form $(A'|\mathbf{b}')$ with Gauss-Jordan reduction, and choosing vectors in the hyperplane by using this matrix, as summarized below:

Algorithm 4.2 - Initialising particles within the search plane

1. Reduce the augmented matrix $(A|\mathbf{b})$ to transform the coefficient matrix A of the given constraints to row-echelon form. The number of pivots in this form will be equal to r, the rank of A.

$$(A|\mathbf{b}) \sim (A'|\mathbf{b}') = \left(\begin{array}{cccccccc|c} 1 & 0 & \ldots & 0 & a'_{1r+1} & \ldots & a'_{1n} & b'_1 \\ 0 & 1 & \ldots & 0 & a'_{2r+1} & \ldots & a'_{2n} & b'_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & a'_{rr+1} & \ldots & a'_{rn} & b'_r \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \ldots & 0 & 0 & \ldots & 0 & 0 \end{array}\right)$$

   (No pivots appear after column r)

2. Use $[A'|\mathbf{b}']$ to generate a total of $n-r$ linearly independent random vectors such that $A\mathbf{p}_i^{(0)} = \mathbf{b}$ for $i = 1 \ldots n-r$.

   A random vector $\mathbf{p} = (p_1, p_2, \ldots, p_n)^T$ satisfying $A\mathbf{p} = \mathbf{b}$ can be constructed by choosing values for p_k randomly, with $k = r+1, \ldots, n$ ($k \in$ *non-pivot columns*). Now for each $j = 1, \ldots, r$ ($j \in$ *pivot columns*), let

$$p_j = b'_j - \sum_{k=r+1}^{n} a'_{jk} p_k$$

3. Generate one more vector $\mathbf{p}_{n-r+1}^{(0)} = \sum_{i=1}^{n-r} \frac{1}{n-r} \mathbf{p}_i^{(0)}$. Now,

$$A\mathbf{p}_{n-r+1}^{(0)} = A \sum_{i=1}^{n-r} \frac{1}{n-r} \mathbf{p}_i^{(0)} = \sum_{i=1}^{n-r} \frac{1}{n-r} A\mathbf{p}_i^{(0)} = \sum_{i=1}^{n-r} \frac{1}{n-r} \mathbf{b} = \mathbf{b}$$

   This vector is a combination of all other vectors and is linearly dependent on all vectors $1, \ldots, n-r$. Any $S^{(0)} - \hat{\mathbf{z}}^{(0)}$ will form a basis for $\mathbb{R}^{n-r}$, since subtracting any choice of $\hat{\mathbf{z}}^{(0)}$ will give a linearly independent set.

4. Choose the initial positions of particles $n-r+2$ to s at random by using the method described in Step 2 to create a swarm of size s.

The computational cost for Gauss-Jordan reduction is $\mathcal{O}(n^3)$, while the cost for back-substitution is $\mathcal{O}(n)$ [8]. Since back-substitution is done a constant number of times (once for each particle), the worst-case complexity for initialising particles in the search plane is therefore for $\mathcal{O}(n^3)$.

4.3.3 Overcoming premature convergence

The LPSO algorithm (Algorithm 4.1) discussed above has one property that is very disadvantageous, and that is the possibility of premature convergence.

If $\mathbf{v}^{(0)}$ is initialised to $\mathbf{0}$ and the position of the global best particle does not change, searches will continue on lines connecting each particle with the global best. So the whole hyperplane is not searched, but only lines.

In another scenario, consider $\mathbf{p}_i = \mathbf{z}_i = \hat{\mathbf{z}}$, where velocity updates will depend only on the value of $w\mathbf{v}_i^{(t)}$, as discussed in [55, 56]. If a particle's current position coincides with the global best position, the particle will only move away from this point if its previous velocity and w are non-zero. Premature convergence will occur when previous velocities are close to zero, and particles stop moving once they catch up with the global best particle.

To overcome this premature convergence, the Guaranteed Convergence Particle Swarm Optimiser (GCPSO) was developed [55, 56]. In this algorithm, the velocity update for the

global best particle is changed to force it to search for a better solution in an area around the position of that particle. A convergence proof for the GCPSO, and results to substantiate its success can be found in [55, 56].

The GCPSO cannot be used as given in [55, 56], since unconstrained random adjustments may generate infeasible solutions. A variation is necessary because particles cannot be altered with any random vector, but only with feasible directions. The new algorithm, referred to as Converging LPSO (CLPSO), ensures that the constraints from equation (4.26) are still met.

Let τ be the index of the global best particle, then

$$\mathbf{z}_\tau = \hat{\mathbf{z}} \tag{4.28}$$

Change the velocity update equation (4.20) for the global best particle τ, so that

$$\mathbf{v}_\tau^{(t+1)} = -\mathbf{p}_\tau^{(t)} + \hat{\mathbf{z}}^{(t)} + \rho^{(t)} \boldsymbol{\upsilon}^{(t)} \tag{4.29}$$

where $\rho^{(t)}$ is a scaling factor and $\boldsymbol{\upsilon}^{(t)} \sim UNIF(-1, 1)^n$ with the property that $A\boldsymbol{\upsilon}^{(t)} = \mathbf{0}$, or $\boldsymbol{\upsilon}^{(t)}$ lies in the null space of A. The vector $\boldsymbol{\upsilon}^{(t)}$ can be constructed from the reduced augmented matrix $[A'|\mathbf{b}']$, with A in row-echelon form. Such a method is described in Step 2 of Section 4.3.2. Now,

$$\begin{aligned} A\mathbf{v}_\tau^{(t+1)} &= A\big(-\mathbf{p}_\tau^{(t)} + \hat{\mathbf{z}}^{(t)} + \rho^{(t)}\boldsymbol{\upsilon}^{(t)}\big) \\ &= -A\mathbf{p}_\tau^{(t)} + A\hat{\mathbf{z}}^{(t)} + \rho^{(t)} A\boldsymbol{\upsilon}^{(t)} \\ &= -\mathbf{b} + \mathbf{b} + \mathbf{0} \\ &= \mathbf{0} \end{aligned}$$

and so the swarm will still fly through the hyperplane as described in Theorem 4.2. Since

$$\begin{aligned} \mathbf{p}_\tau^{(t+1)} &= \mathbf{v}_\tau^{(t+1)} + \mathbf{p}_\tau^{(t)} \\ &= \hat{\mathbf{z}}^{(t)} + \rho^{(t)}\boldsymbol{\upsilon}^{(t)} \end{aligned}$$

the new position of the global best particle will be its personal best $\hat{\mathbf{z}}^{(t)}$, with a random vector $\rho^{(t)}\boldsymbol{\upsilon}^{(t)}$ from the null space of A added. It is *only* the global best particle that is moved with the above velocity update (4.29), all other particles in the swarm are still moved with the original equations (4.20) and (4.21).

Removal of initial conditions for CLPSO

Adding random vectors to the algorithm changes the initial conditions: Theorem 4.1 is based on LPSO which does not make any allowance for random changes to particle positions. Since $\boldsymbol{\upsilon}^{(t)}$ is random, the condition that some $S^\star \subseteq S^{(0)} - \hat{\mathbf{z}}^{(0)}$ that defines a basis for $\mathbb{R}^{n-r}$ (with

$rank(A) = r$) should exist, can be dropped for CLPSO. Algorithm 4.2 can still be used to initialise particles within the search space, but a swarm size of *less* than $n - r$ particles is now possible.

4.3.4 Proof of convergence for CLPSO

To prove the convergence of CLPSO to at least a local minimum (which is a global minimum for a convex function), a more general condition for convergence of a random search algorithm is first discussed and proved. Consider the following problem and conceptual algorithm, adapted from [52]:

P *Given a measurable function* $f : \mathbb{R}^n \to \mathbb{R}$ *and* $\mathcal{S} \subseteq \mathbb{R}^n$. *We seek a point* $\mathbf{x} \in \mathcal{S}$ *which at least finds a local minimum of* f *on* $\mathcal{S}$ *or yields an approximation of a local minimum of* f *on* $\mathcal{S}$.

Algorithm 4.3 - Conceptual algorithm

1. Find $\mathbf{x}^{(0)} \in \mathcal{S}$ and set $k = 0$
2. Generate $\boldsymbol{\xi}^{(k)}$ from $(\mathbb{R}^n, \mathcal{B}, \mu_k)$
3. $\mathbf{x}^{(k+1)} = D(\mathbf{x}^{(k+1)}, \boldsymbol{\xi}^{(k)})$, choose μ_{k+1}, $k := k + 1$, go to step 1

The probability space $(\mathbb{R}^n, \mathcal{B}, \mu_k)$ is such that $\mathcal{B}$ is the σ-field of Borel subsets of $\mathbb{R}^n$, and μ_k is a probability measure on $\mathcal{B}$ such that $\mu_k(\mathbb{R}^n) = 1$. The algorithm starts with an initial solution $\mathbf{x}^{(0)}$, and at each iteration a possible new solution $\boldsymbol{\xi}^{(k)}$ is generated from $(\mathbb{R}^n, \mathcal{B}, \mu_k)$. The function D, explained below, maps $\mathcal{S} \times \mathbb{R}^n$ to $\mathcal{S}$.

It is sufficient to show that if the random search algorithm satisfies two conditions – the *algorithm condition* and the *convergence condition* – then it will at least converge to a local minimum. Each of these condisions are presented below.

Algorithm condition The mapping $D : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S}$ should satisfy $f(D(\mathbf{x}, \boldsymbol{\xi})) \leq f(\mathbf{x})$ and if $\boldsymbol{\xi} \in \mathcal{S}$, then $f(D(\mathbf{x}, \boldsymbol{\xi})) \leq f(\boldsymbol{\xi})$.

Let M_k be the support of μ_k, the smallest closed subset of $\mathbb{R}^n$ with measure of one. Almost all random search algorithms are adaptive, implying that μ_k depends on the solutions $\mathbf{x}^{(0)}, \ldots, \mathbf{x}^{(k-1)}$ generated by the previous iterations of the algorithm. The μ_k are then viewed as conditional probability measures. Let m be the Lebesgue measure of a set. The search method discussed here is a called a *local search method*, which means that the μ_k with bounded support M_k have, for all except a possibly finite k, $m(\mathcal{S} \cap M_k) < m(\mathcal{S})$. Methods called *global search methods* have $m(\mathcal{S} \cap M_k) = m(\mathcal{S})$ for all k.

To avoid having to search for an element in a set of null measure, the search will be for the essential infimum of f. This assures that, for a pathological case like $f(x) = x^2$ for $x \neq 0$, and $f(x) = -1$ for $x = 0$, the true minimum at -1 need not be found, but simply an x for which $f(x)$ is arbitrarily close to zero. Thus the search for the infimum will be replaced by a search for the essential infimum. Define the minimum of f on $\mathcal{S}$ as

$$\alpha = \text{ess inf } f = \sup\{z : f(\mathbf{x}) \geq z \text{ a.e.}\}$$

and assume that α is finite.[1]

Since the nature of the search is for the essential infimum and therefore may preclude the actual minimum, it is necessary to establish convergence to a small region of values surrounding the minimum. Let the *optimality region* for the (global) minimum be defined as

$$R_{\epsilon,0} = \{\mathbf{x} \in \mathcal{S} : f(\mathbf{x}) < \alpha + \epsilon\}$$

Function f has an essential local minimum at $\mathbf{c}_i \in \mathcal{S}$ if there exists an n-dimensional interval $I_i \subseteq \mathcal{S}$ around $\mathbf{c}_i$, such that $f(\mathbf{c}_i) \leq f(\mathbf{x})$ a.e. for all $\mathbf{x} \in I_i$. For each of the (possibly infinite) local minima $\mathbf{c}_i$ with $i \geq 1$, define the optimality region (that is sufficient for the search algorithm to find) as

$$R_{\epsilon,i} = \{\mathbf{x} \in I_i : f(\mathbf{x}) < f(\mathbf{c}_i) + \epsilon\}$$

Now let $R_\epsilon = \bigcup_i R_{\epsilon,i}$ be the optimality region for problem $\mathbf{P}$.

Convergence condition *Sufficient condition for convergence to at least a local minimum (of a local search algorithm): For any $\mathbf{x}^{(k)} \in \mathcal{S}$, there exists a $\gamma > 0$ and a $0 < \eta \leq 1$ such that*

$$\mu_k\big(f(\mathbf{x}^{(k+1)}) \leq f(\mathbf{x}^{(k)}) - \gamma \text{ or } \mathbf{x}^{(k)} \in R_\epsilon\big) \geq \eta \tag{4.30}$$

Proof Take the complement of (4.30) to get

$$\mu_k\big(f(\mathbf{x}^{(k+1)}) > f(\mathbf{x}^{(k)}) - \gamma \text{ and } \mathbf{x}^{(k)} \notin R_\epsilon\big) \leq 1 - \eta$$

for all $\gamma > 0$.

From the definition of D, $f(\mathbf{x}^{(k+1)}) > f(\mathbf{x}^{(k)}) - \gamma$ for all $\gamma > 0$ is not possible, and so

$$\mu_k(\mathbf{x}^{(k)} \notin R_\epsilon) \leq 1 - \eta$$

[1] Thus the minimum is defined as the supremum of all z values such that f is greater than or equal to z *almost everywhere* (a.e.), i.e. everywhere except possibly on some null set. Letting $\alpha = -\infty$ will not alter the spirit of the algorithm, if a very large negative value is taken as a sufficient 'approximation' of $-\infty$.

Let $\{\mathbf{x}^{(k)}\}_{k\geq 0}$ be the sequence generated by D. Therefore it needs to be shown that $\lim_{k\to\infty} P(\mathbf{x}^{(k)} \in R_\epsilon) = 1$. Define A to be the event that $\mathbf{x}^{(k)} \in R_\epsilon$ before iteration p. Then,

$$\begin{aligned} P(A) &= 1 - P(\bar{A}) \\ &= 1 - \prod_{i=0}^{p-1} \mu_i(\mathbf{x}^{(i)} \notin R_\epsilon) \\ &\geq 1 - (1-\eta)^p \end{aligned}$$

and so $P(A) \to 1$ as $p \to +\infty$.

To complete the proof, consider the case when $\mathbf{x}^{(p)} \in R_\epsilon$, and $\mu_p(f(\mathbf{x}^{(p+1)}) \leq f(\mathbf{x}^{(p)}) - \gamma) > 0$. Then there is a positive probability that $\mathbf{x}^{(p+1)} \notin R_\epsilon$, and if that is the case, the above argument assures us that $\mathbf{x}^{(k)}$ will converge to R_ϵ once again. From the definition of R_ϵ and D, this will be to a local or possibly global minimum less than $\mathbf{x}^{(p)}$. (When $\mu_p(f(\mathbf{x}^{(p+1)}) \leq f(\mathbf{x}^{(p)}) - \gamma) = 0$, the sequence will remain in R_ϵ at a local or the global minimum.) □

To prove that CLPSO converges at least to a local minimum, and does not stagnate and converge prematurely, it needs to be shown that both the *algorithm condition* and the *convergence condition* defined above will hold. Let $\mathcal{S} = \mathbb{R}^n$.

Algorithm condition The global best $\widehat{\mathbf{z}}^{(t)}$ is set to the position of the best performance in the swarm, i.e.

$$\begin{aligned} \widehat{\mathbf{z}}^{(t)} \in \{\mathbf{z}_1^{(t)}, \mathbf{z}_2^{(t)}, \ldots, \mathbf{z}_s^{(t)}\} \mid f(\widehat{\mathbf{z}}^{(t)}) \\ = \max\{f(\mathbf{z}_1^{(t)}), f(\mathbf{z}_2^{(t)}), \ldots, f(\mathbf{z}_s^{(t)})\} \end{aligned}$$

and

$$\mathbf{z}_i^{(t)} = \begin{cases} \mathbf{z}_i^{(t-1)} & \text{if } f(\mathbf{p}_i^{(t)}) \geq f(\mathbf{z}_i^{(t-1)}) \\ \mathbf{p}_i^{(t)} & \text{if } f(\mathbf{p}_i^{(t)}) < f(\mathbf{z}_i^{(t-1)}) \end{cases}$$

The above update equations imply that the *algorithm condition* holds.

Convergence condition Particle update equations are

$$\mathbf{p}_i^{(t+1)} = \mathbf{p}_i^{(t)} + w\mathbf{v}_i^{(t)} + c_1 r_1^{(t)}[\mathbf{z}_i^{(t)} - \mathbf{p}_i^{(t)}] + c_2 r_2^{(t)}[\widehat{\mathbf{z}}^{(t)} - \mathbf{p}_i^{(t)}]$$

and for the global best particle

$$\mathbf{p}_\tau^{(t+1)} = \widehat{\mathbf{z}}^{(t)} + \rho^{(t)}\boldsymbol{\upsilon}^{(t)}$$

Sampling a new point (that might be better than $\widehat{\mathbf{z}}^{(t)}$) will be done for each of s particles, and thus we will define M_t, the support for μ_t at iteration t, as the set from which each

of these s values can be picked. For each particle $\mathbf{p}_i$ (except for the global best particle τ) define $M_{t,i}$ as the convex hull defined by $(\mathbf{p}_i^{(t)})$, $(\mathbf{p}_i^{(t)} + w\mathbf{v}_i^{(t)})$, $(\mathbf{p}_i^{(t)} + c_1[\mathbf{z}_i^{(t)} - \mathbf{p}_i^{(t)}])$, and $(\mathbf{p}_i^{(t)} + c_2[\widehat{\mathbf{z}}^{(t)} - \mathbf{p}_i^{(t)}])$.

Since $r_1^{(t)}, r_2^{(t)} \sim UNIF(0,1)$, the new particle $\mathbf{p}_i^{(t+1)}$ will lie within $M_{t,i}$. Also define $M_{t,\tau}$ as the n-dimensional hypercube with sides of length $\rho^{(t)}$, centered at $\widehat{\mathbf{z}}^{(t)}$. Let

$$M_t = \bigcup_{i=1}^{s} M_{t,i}$$

be the support of probability measure μ_t. Since $M_{t,\tau} \subseteq M_t$ a point arbitrarily close to $\widehat{\mathbf{z}}^{(t)}$ can be chosen, and hence there is always a $\gamma > 0$ and $0 < \eta \leq 1$ such that

$$\mu_t\big(f(\widehat{\mathbf{z}}^{(t+1)}) \leq f(\widehat{\mathbf{z}}^{(t)}) - \gamma \text{ or } \widehat{\mathbf{z}}^{(t)} \in R_\epsilon\big) \geq \eta$$

□

4.4 Inequality-constrained optimisation

Inequality-constrained optimisation problems can be reduced to problems involving only non-negativity constraints on a set of variables. In Section 4.1.3 the notion of slack variables, where a standard optimisation problem is converted to one where all inequalities involve only a single variable, was introduced. The LPSO, and consequently the CLPSO as well, are expanded to handle non-negativity constraints on a set of variables. As the aim of the CLPSO is (in the context of this thesis) to solve a SVM's constrained optimisation problem, the method explained below focuses on box constraints of the form $a \leq x_j \leq b$. These constraints force the particles to only fly inside a n-dimensional hypercube, but the method developed will work equally well if no upper bound on the variables existed.

Consider the way a particle $\mathbf{p}_i$ is being updated:

$$\mathbf{p}_i^{(t+1)} = \mathbf{v}_i^{(t+1)} + \mathbf{p}_i^{(t)} \tag{4.31}$$

In the above equation, it is also assumed that $\mathbf{p}_i^{(t)}$ lies inside the problem's feasible region Ω. That is, inside the n-dimensional hypercube. For notational convenience, the subscript i will be dropped. That is,

$$\mathbf{p}^{(t)} = (p_1^{(t)}, p_2^{(t)}, \ldots, p_n^{(t)})^T \tag{4.32}$$

For the above particle, for all values $p_j^{(t)}$ it will be true that $a \leq p_j^{(t)} \leq b$. However, when the velocity vector $\mathbf{v}^{(t+1)}$ is added, it may become true that a value of $p_j^{(t+1)}$ may violate these constraints.

In this case, the velocity vector needs to be scaled so that all values $p_j^{(t+1)}$ will fall inside the constraints. To scale the velocity vector, a scale factor is computed for each $p_j^{(t+1)}$ that lies outside of the constraints. This factor will scale the vector element to lie exactly on the bound. Since the scale factor of one element may scale other elements to lie outside of the bounds, the minimum of all these scale factors are taken to scale the velocity vector. Using this simple technique, the movement of the particles are restricted to the hypercube.

As an example, let $a = 0$ and $b = 2$ such that $0 \leq p_j^{(t)} \leq 2$ in the following position vector, and consider the addition of a velocity vector:

$$\begin{aligned}
\mathbf{p}^{(t)} &= (\ \tfrac{1}{8}\ \tfrac{1}{8}\ \tfrac{6}{8}\ 0\ 0\ \tfrac{7}{8}\ \tfrac{1}{8}\ )^T \\
\mathbf{v}^{(t+1)} &= (\ 0\ 0\ -\tfrac{8}{8}\ 0\ 0\ \tfrac{10}{8}\ \tfrac{18}{8}\ )^T \\
\mathbf{p}^{(t+1)} &= (\ \tfrac{1}{8}\ \tfrac{1}{8}\ -\tfrac{2}{8}\ 0\ 0\ \tfrac{17}{8}\ \tfrac{19}{8}\ )^T \\
&\qquad\quad < 0 \qquad > 2 > 2
\end{aligned}$$

It is clear that the new particle lies outside the $[0, 2]^7$ hypercube. For scaling, a value δ needs to be found such that $\mathbf{p}^{(t+1)} = \delta \mathbf{v}^{(t+1)} + \mathbf{p}^{(t)}$ will lie inside these constraints. This δ must be chosen such that $p_3^{(t+1)}$, which is smaller than $a = 0$, will now satisfy $p_3^{(t+1)} \geq 0$. The value of δ must also enforce $p_6^{(t+1)} \leq 2$ and $p_7^{(t+1)} \leq 2$.

Continuing the example, δ is computed for each violating dimension. The value of $p_3^{(t+1)}$ is $-\frac{2}{8}$, but it should ideally be 'cut' to lie within its closest boundary, zero. Substituting zero for $p_3^{(t+1)}$ gives the scaling factor δ with which the velocity vector should be scaled to achieve this ideal value:

$$\begin{aligned}
p_3^{(t+1)} &= \delta_3 v_3^{(t+1)} + p_3^{(t)} \\
\delta_3 &= (p_3^{(t+1)} - p_3^{(t)})/v_3^{(t+1)} \\
&= (0 - \frac{6}{8})/(-\frac{8}{8}) = \frac{6}{8}
\end{aligned} \tag{4.33}$$

Similarly, the value for $p_6^{(t+1)}$ is $\frac{17}{8}$, but should ideally be scaled down to two, to lie within its closest border:

$$\begin{aligned}
\delta_6 &= (p_6^{(t+1)} - p_6^{(t)})/v_6^{(t+1)} \\
&= (2 - \frac{7}{8})/(\frac{10}{8}) = \frac{9}{10}
\end{aligned} \tag{4.34}$$

The value for $p_7^{(t+1)}$ is $\frac{19}{8}$, but should also be scaled down to two, to lie within its closest border:

$$\begin{aligned}
\delta_7 &= (p_7^{(t+1)} - p_7^{(t)})/v_7^{(t+1)} \\
\delta_7 &= (2 - \frac{1}{8})\ (\frac{18}{8}) = \frac{15}{18}
\end{aligned} \tag{4.35}$$

From these possible scale values that were computed in (4.33), (4.34), and (4.35), the smallest δ is chosen to scale the velocity vector with. Thus the value of δ will be $\frac{6}{8}$. Multiplying δ

with $\mathbf{v}^{(t+1)}$ and updating the particle gives a new position $\mathbf{p}^{(t+1)}$ that lies exactly within the constraints.

$$\begin{aligned}
\mathbf{p}^{(t)} &= (\ \tfrac{1}{8}\ \tfrac{1}{8}\ \tfrac{6}{8}\ 0\ 0\ \tfrac{7}{8}\ \tfrac{1}{8}\ )^T \\
\delta\mathbf{v}^{(t+1)} &= (\ 0\ 0\ -\tfrac{6}{8}\ 0\ 0\ \tfrac{15}{16}\ \tfrac{27}{16}\ )^T \\
\mathbf{p}^{(t+1)} &= (\ \tfrac{1}{8}\ \tfrac{1}{8}\ 0\ 0\ 0\ \tfrac{29}{16}\ \tfrac{29}{16}\ )^T
\end{aligned}$$

From the above example, an algorithm to keep a swarm of particles within an n-dimensional hypercube $[a, b]^n$, can be generalised.

Algorithm 4.4 - Satisfying inequality constraints

1. Determine the new position that a particle will fly to (but do not move it there)

$$\mathbf{p}^{(t+1)} = \mathbf{v}^{(t+1)} + \mathbf{p}^{(t)}$$

2. For each dimension j in the new position that lies outside $[a, b]^n$, compute a scaling factor δ_j

$$\begin{aligned}
\delta_j &= \left(a - p_j^{(t)}\right)/v_j^{(t+1)} \text{ if } p_j^{(t+1)} < a \\
\delta_j &= \left(b - p_j^{(t)}\right)/v_j^{(t+1)} \text{ if } p_j^{(t+1)} > b
\end{aligned}$$

   Note that, since $p_j^{(t)} \in [a, b]$ and $p_j^{(t+1)} \notin [a, b]$, the value of δ will always be positive.

3. Set $\delta = \min\{\delta_j \mid p_j^{(t+1)} \notin [a, b]\}$

4. Finally, move the particle to the new position with

$$\mathbf{p}^{(t+1)} = \delta\mathbf{v}^{(t+1)} + \mathbf{p}^{(t)}$$

   to lie within the constrained hypercube $[a, b]^n$.

If each dimension of a particle lies outside $[a, b]^n$, it takes at most n divisions to find the scaling factors δ_j, and a maximum of n comparisons to find δ, giving an $\mathcal{O}(n)$ complexity. The method described above in Algorithm 4.3 is used and experimentally verified as part of the CLPSO used for training Support Vector Machines.

It is now possible to fly the swarm such that both linear and bounded constraints are always met. However, the above approach of 'cutting against the borders' induces a new hurdle that the LPSO has to overcome.

The LPSO requires that the set of vectors created by subtracting the particle's current position from the global best solution vector, together with the swarm's set of velocity vectors, must span the entire search space. If all particles are 'cut' against a single constraint (say a in $a \leq p_j \leq b$, as shown in Figure 4.4), the particle positions may all become

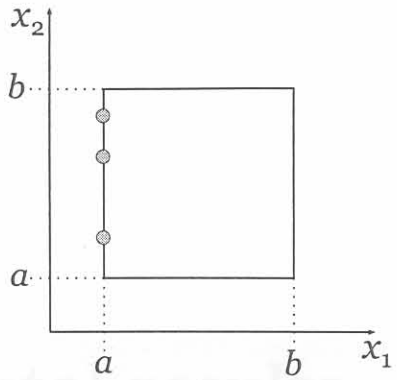


FIGURE 4.4: Particles becoming a linear combination of each other.

linear combinations of each other, and if the global best also lies on the specific constraint, the property of spanning the search space will be lost. This problem can be remedied by randomly scattering the swarm, or adding a random vector to each particle to move its current position to the inside of the box constraints, when no improvement is made in the objective function for a fixed number of iterations.

Due to the way the global best particle is moved in CLPSO, a random vector is always added to a position in the swarm. The random vector ensures that, with a probability greater than zero for each iteration, that the global best particle will be moved away from the bound to be inside (a, b).

4.5 Concluding

In this chapter the original form of the PSO algorithm was extended to solving constrained optimisation problems. Two new PSO algorithms were developed. The Linear PSO (LPSO) makes it possible to traverse a search space as a hyperplane, and conditions for LPSO to reach any point within the search space were rigorously analysed. LPSO does however make allowance for premature convergence. To remedy the problem of premature convergence, the Converging LPSO (CLPSO) was developed. A formal proof of CLPSO convergence was given. Finally, a method of handling inequality (box) constraints was presented.

Experimental results follow in the next chapter, and illustrate LPSO and CLPSO on a number of problems, as well as their performance as an optimiser in Support Vector Machine training.