Agent-Based Problem Solving Using Differential Evolution on Real-Valued Problems

Loizos Kounios
Intercollege-Larnaca
Cyprus
loizoskounios@gmail.com

Abstract

Differential Evolution (DE) is a powerful agent-based stochastic solution optimizer for real-valued problems. Like many *Evolutionary Algorithms*, it is based upon ideas taken from Darwinian genetics such as survival of the fittest, parents crossing "genes" to produce robust children, and genetic mutations.

This paper presents:

- An introduction to Differential Evolution
- A detailed look into the Differential Evolution algorithm and concepts
- A report on the Differential Evolution Toolkit (DET) test results
- An analysis on how the scaling factor and crossover rate parameters affect the convergence rate of Differential Evolution
- An introduction and performance analysis of the *DE/best-to-next/1* mutation scheme

Graphs and tables are supplied to further aid the understanding of the convergence rate for different types of problems.

The results are impressive and show the power of DE at optimizing difficult problems. Moreover, the overall performance improvement the *DE/best-to-next/1* mutation scheme has yielded is proof that even though DE is an extremely powerful tool for mathematical optimization, modifications to the algorithm provide even better results.

Contents

Chapter 1

Introduction

Mathematical optimization is a large, complex "open problem." For decades, scientists have been attempting to find methods which will provide a reduction in the scope of the issue. The purpose of optimization is to improve what currently exists and make it adapt as well as possible to its new and ever-changing environment. When faced with this problem, scientists decided to look at nature for a solution—after all, nature has been optimizing in the sense of having species adapt to their environment for eons now. Those who managed to adapt have survived; those who failed to adapt have gone extinct.

Darwinian evolution and its "survival of the fittest" paradigm were the inspiration for *Evolutionary Computation* (EC), the umbrella term for all biologically inspired computing. One of the many evolutionary algorithms that was developed using nature as a model was *Differential Evolution*.

The first published work on Differential Evolution (DE) was by R. Storn and K. V. Price (University of California, Berkeley) in 1995[1]. In May 1996, DE took place in the First International Conference on Evolutionary Computation (CEC) and the results were very promising—it finished third having only lost to two non-evolutionary and non-universally applicable algorithms[2], which made DE the best performing *evolutionary* algorithm that participated in the conference.

Differential Evolution (DE) is a powerful agent-based stochastic solution optimizer for real-valued problems. DE optimizes a function by iteratively attempting to improve a candidate solution. DE is based on ideas taken from Darwinian evolution and, more specifically, genetics. Like many evolutionary algorithms, DE attempts to simulate the natural process of evolution by using ideas such as genetic mutation, parents "crossing genes" to produce offspring, and survival of the fittest.

DE has been successfully used in a variety of different problems (image processing[3;4], scheduling[5], optimal design[6], community detection[7] etc.) since its inception in 1995. DE provides no magic formula and even though it is labeled as an algorithm, it does not satisfy the technical properties of one. DE is, in fact, a computational method (or heuristic) that might not always find a solution, but will at the very least provide the user with information about where to look for one. DE performs extremely well against problems that are discrete, discontinuous, temporally changeable or highly noisy. DE remains an active research topic in the *Evolutionary Computing* (EC) community.

In February 2011, Das and Suganthan published a literature review in the IEEE Transactions on Evolutionary Computation journal that contained state-of-the-art DE techniques and eight "future research directions" on DE[8]. Of the potential research directions listed, items 5 and 6 (discussed in Section 2.11) are particularly interesting for improving DE.

The rest of the report is arranged as follows: in Chapter 2, the basic concepts and algorithm of Differential Evolution are explained; in Chapter 3, the *Differential Evolution Toolkit* (DET) developed is tested against problems from the "CEC 2005 Test Suite[9];" in Chapter 4, the *DE/best-to-next/1* mutation scheme is tested, and lastly Chapter 5 concludes the report.

Bibliography

[1] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *ICSI, USA*, 1995.

[2] R. Storn and K. Price, "Minimizing the real functions of the icec 1996 contest by differential evolution," *Proc. IEEE Int. Conf. Evol. Comput.*, pp. 842–844, 1996.

[3] I. D. Falco, A. D. Cioppa, and A. Tarantino, "Automatic classification of handsegmented image parts with differential evolution," *EvoWorkshops 2006*, pp. 403–414, 2006.

[4] M. Omran, A. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," *Proceedings of IEEE Congress on Evolutionary Computation*, vol. 2, pp. 966–973, 2005.

[5] A. Nearchou and S. Omirou, "Differential evolution for sequencing and scheduling optimization," *Journal of Heuristics*, no. 12, pp. 395–411, 2006.

[6] B. Babu and S. Munawar, "Differential evolution strategies for optimal design of shell-and-tube heat exchangers," *Chemical Engineering Science*, vol. 62, no. 14, 2007.

[7] G. Jia, Z. Cai, M. Musolesi, Y. Wang, D. Tennant, R. Weber, J. Heath, and S. He, "Community detection in social and biological networks using differential evolution," *Learning and Intelligent OptimizatioN Conference*, 2012.

[8] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[9] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," 2005.

Chapter 2

Differential Evolution Concepts

DE is an evolutionary computation technique that deals with the problem of search and optimization. DE searches for a global optimum in regards to the requirements of a problem—minimization or maximization of an *objective function*.

DE consists of four stages: (1) *initialization* of the population, (2) *mutation*, (3) *crossover*, and (4) *selection*. Initialization is the first step of the algorithm, after which the algorithm is repeatedly going through the mutation-crossover-selection cycle, attempting to improve on a *candidate solution* until an ending criterion has been satisfied. The stages of DE can be seen in Figure 2.1.

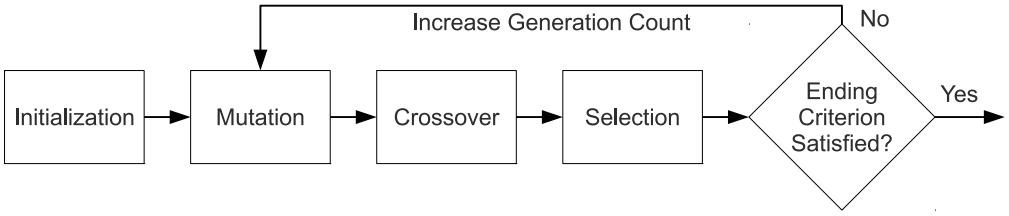


Figure 2.1: The stages of DE.

What makes DE an extremely attractive optimization technique is the fact that its performance is dictated by only three control parameters: the population size, NP, the scaling factor, F, and the crossover rate, Cr. During the early years of DE, it was thought that the choice of control parameters was an easy task[1;2]. Research has shown, however, that the performance of the algorithm is extremely sensitive to the choice of control parameters[3–6].

The DE parameters are usually represented in vector notation as shown in Eq 2.1.

$$\vec{X} = [x_1, x_2, x_3, \ldots, x_D]^T \tag{2.1}$$

where x_i is any real number in the D-dimensional real parameter search space $\mathbb{R}^D$. In DE, there exist three kinds of vectors: a parent vector—referred to as *target* vector, $\vec{X}$; a mutant vector which is obtained through the mutation operation—referred to as *donor* vector, $\vec{V}$; and an offspring vector that is the result of the recombination of the target and donor vectors—referred to as *trial* vector, $\vec{U}$. All three vectors are 2-dimensional and of size $NP \times D$ as can be seen in Figure 2.2.

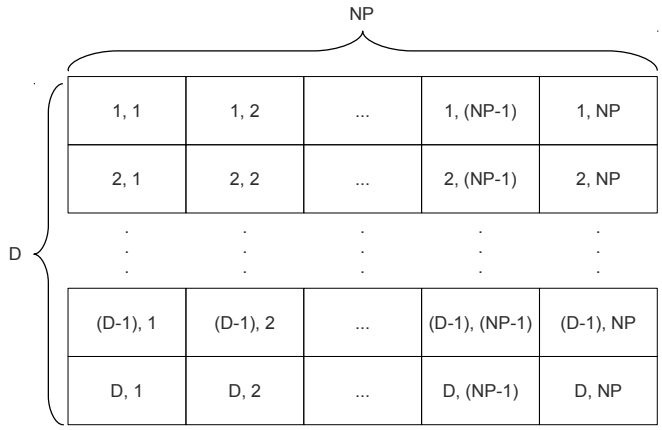


Figure 2.2: A visual representation of a DE vector.

During the optimization of a problem, the DE algorithm will be trying to find the parameter vector, $\vec{X}^*$, that provides the best candidate solution for the objective function in the *search space* provided. This can be best expressed as Equation 2.2a for the minimization and Equation 2.2b for the maximization of an objective function.

$$f(\vec{X}^*) < f(\vec{X}) \text{ for all } \vec{X} \in \Omega \tag{2.2a}$$

$$f(\vec{X}^*) > f(\vec{X}) \text{ for all } \vec{X} \in \Omega \tag{2.2b}$$

where Ω is the domain of the search.

The following sections discuss thoroughly the stages and concepts of DE. Specifically, the first four sections refer to the main stages of the DE algorithm: Section 2.1 refers to the initialization stage, Section 2.2 to the mutation stage, Section 2.3 to the crossover stage and Section 2.4 to the selection stage. After the basic DE stages are discussed, the objective function (Section 2.5), objective result (Section 2.6) and terminating conditions (Section 2.7) are defined. Everything discussed in the previous sections is put together in the form of the classic DE algorithm in Section 2.8. Section 2.9 lists the most prominent variants of DE while Section 2.10 and Section 2.11 attempt to address the drawbacks of DE and list future research directions for the algorithm. Lastly, a summary of all that was discussed is provided in Section 2.12.

2.1 Initialization

Initialization takes place at "Generation Zero" and instantiates the NP D-dimensional population with uniformly generated random numbers in a pre-defined search space.

The initial population is uniformly initialized in the range of each dimension's minimum and maximum constraints, i.e., the search space. $\vec{X}_{min}$ and $\vec{X}_{max}$ store the minimum and maximum constraints per dimension and can be shown as:

$$\vec{X}_{min} = \{x_{1,min}, x_{2,min}, \ldots, x_{D,min}\}$$

$$\vec{X}_{max} = \{x_{1,max}, x_{2,max}, \ldots, x_{D,max}\}$$

The following notation is used for referring to a vector at a particular generation:

$$\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots, x_{D,i,G}] \tag{2.3}$$

where i lies in the range of $[1, \mathrm{NP}]$ and G is the generation number. Subsequent generations are denoted by $G = [0, 1, \ldots, G_{max}]$.

Therefore, the initialization of the j^{th} element of the i^{th} vector at "Generation Zero" is achieved using the following equation:

$$x_{j,i,0} = x_{j,min} + \mathrm{rand}[0,1] \cdot (x_{j,max} - x_{j,min}) \tag{2.4}$$

2.1.1 Literature on Initialization

Storn stressed that the initialization stage is very important and that the initialization of the target vector should be as uniform as possible[7]. Storn and Price have suggested that the size of the population, NP, should range from $(5 \cdot D)$ to $(10 \cdot D)$, with D being the amount of dimensions of a problem[8] while Gämperle et al. counter-claim[4] that the ideal range for NP is $(3 \cdot D)$ to $(8 \cdot D)$. Rönkkönen et al. supported that a suitable value for NP would be between $(2 \cdot D)$ and $(40 \cdot D)$. Rönkkönen et al. also pointed out that the larger the population size, the greater the chance of successful convergence, but noted that an overly large population may needlessly increase the computational time because of extra function evaluations[5].

2.2 Mutation

Biologically speaking, when it is said that something is "mutated," it means that the gene characteristics of its chromosome were suddenly altered. In Evolutionary Computation, however, mutation refers to the perturbation of an element with another element. The mutation process is where the donor vector is created and populated.

In order for mutation to work, one donor vector, $\vec{V}_{i,G}$, needs to be created for every target vector, $\vec{X}_{i,G}$. Once the donor vector has been created, the target vector is "mutated" (by going through the mutation operation) and copied to the corresponding donor vector. The mutation scheme describes the way in which the data are "mutated."

2.2.1 Mutation Schemes

A mutation scheme is a technique used for mutating the target vector. Mutation schemes are distinct in the sense of the amount of difference vectors and choice of elements considered for perturbation. Some of the most common mutation schemes are listed below and are accompanied by visuals (Figure 2.3) and an example code of the classic mutation scheme "DE/rand/1" (Algorithm 1).

$$\text{"DE/rand/1"}: \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) \tag{2.5a}$$

$$\text{"DE/best/1"}: \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) \tag{2.5b}$$

$$\text{"DE/target-to-best/1"}: \vec{V}_{i,G} = \vec{X}_{i,G} + F \cdot (\vec{X}_{best,G} - \vec{X}_{i,G}) + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) \tag{2.5c}$$

$$\text{"DE/rand/2"}: \vec{V}_{i,G} = \vec{X}_{r_1^i,G} + F \cdot (\vec{X}_{r_2^i,G} - \vec{X}_{r_3^i,G}) + F \cdot (\vec{X}_{r_4^i,G} - \vec{X}_{r_5^i,G}) \tag{2.5d}$$

$$\text{"DE/best/2"}: \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{r_1^i,G} - \vec{X}_{r_2^i,G}) + F \cdot (\vec{X}_{r_3^i,G} - \vec{X}_{r_4^i,G}) \tag{2.5e}$$

The indices r_1^i, r_2^i, r_3^i, r_4^i, and r_5^i are mutually exclusive integers randomly chosen from the range $[1, \mathrm{NP}]$, and all are different than the base vector index i to guarantee that the mutated vector is a mutant of different vectors and not of itself. This set of indices is randomly generated once per donor vector. The *scaling factor*, F, is a positive control parameter in the range of $[0, 1]$ for scaling the difference vector. In the case of DE families that take the "best" vector into account in the mutation equation, $\vec{X}_{best,G}$ is the best performing vector (i.e., when minimizing a function, the vector that returns the lowest result) in the population at generation G.

Algorithm 1 Mutation function using DE/rand/1.

for $i = 1 \rightarrow NP$ **do**

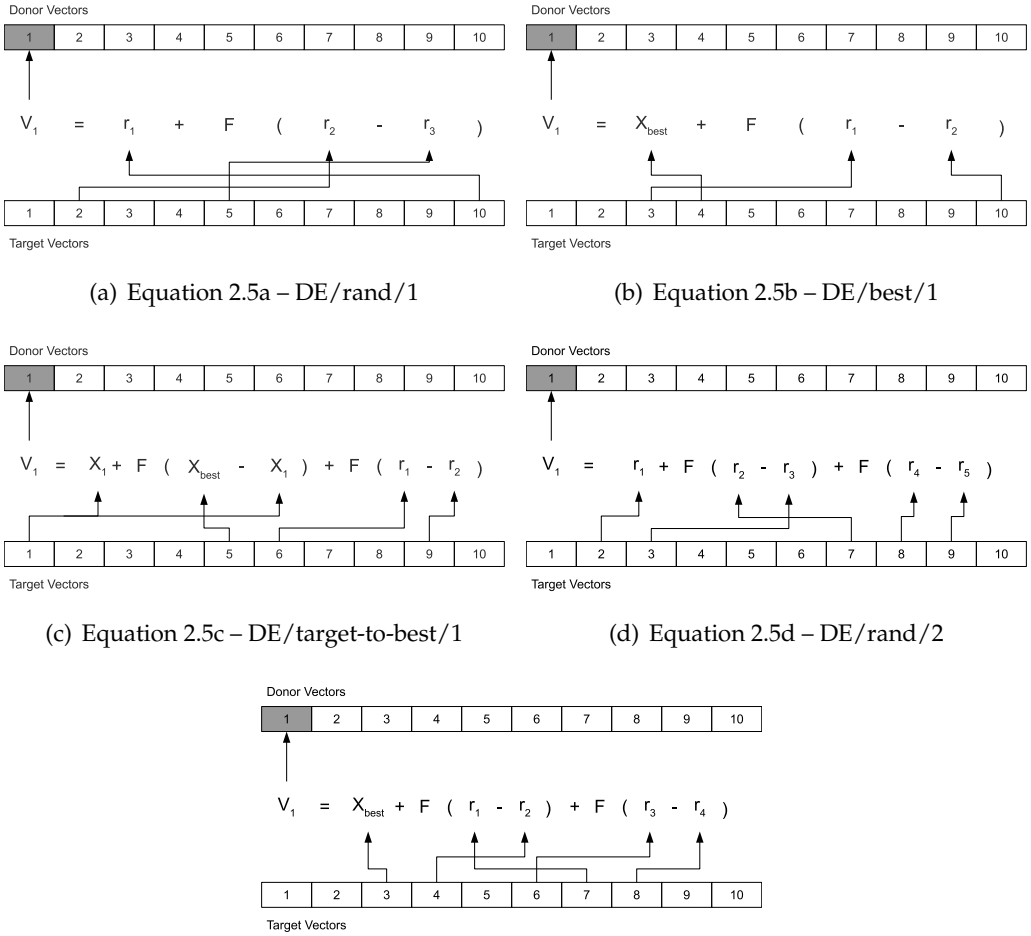
$x[3] = getRandomIndices()$ ▷ Returns three mutually exclusive random numbers that are also different from i.

for $j = 1 \rightarrow D$ **do**

$donorVector[i][j] = targetVector[x[1]][j] + F * (targetVector[x[2]][j] - targetVector[x[3]][j])$

end for

end for



(a) Equation 2.5a – DE/rand/1

(b) Equation 2.5b – DE/best/1

(c) Equation 2.5c – DE/target-to-best/1

(d) Equation 2.5d – DE/rand/2

(e) Equation 2.5e – DE/best/2

Figure 2.3: A visual representation of the mutation schemes when mutating index 1.

2.2.2 Mutation Variants

The names for the mutation schemes follow a simple layout: "DE/x/y" where "DE" stands for "Differential Evolution," "x" refers to the base vector or vectors considered for perturbation and "y" is the amount of difference vectors considered for perturbation. For instance, the "DE/rand/1" mutation scheme translates to a Differential Evolution algorithm that perturbs the target vector using a random vector as its base and one difference vector.

2.2.3 Literature on Mutation

Generally, it is agreed that the choice of F is extremely important[4;5]—F should never be lower than a particular value (depending on the difficulty of the objective function and the DE strategy used); otherwise the algorithm converges prematurely and has a lower chance of escaping local optima. It is worth noting that it has also been reported that when $F > 1$, a decrease in convergence speed can be observed.

In April 2011, Jeyakumar and Shanmugavelayutham published a paper on the best performing DE variants[6] and concluded that DE/rand/1/bin, DE/rand/2/bin and DE/best/2/bin had the best performance amongst their peers and converged faster to the solution. "bin" refers to the binomial crossover method which is discussed in detail in the following section.

2.3 Crossover

Upon completion of the mutation operation, the crossover operation takes place to guarantee that the population is sufficiently diversified. Maintaining the diversification of the population is achieved by forming a trial vector
$\vec{U}_{i,G} = [u_{1,i,G}, u_{2,i,G}, u_{3,i,G}, \ldots, u_{D,i,G}]$, and populating it with elements from the target vector, $\vec{X}_{i,G}$, and the donor vector, $\vec{V}_{i,G}$. The most commonly used crossover methods are *exponential* (exp) and *binomial* (bin). Both methods are controlled by a control parameter named *crossover rate*, or Cr.

2.3.1 Exponential Crossover

In exponential crossover, the donor vector contributes a series of consecutive elements to the trial vector; the remaining indices are populated with data from the target vector. This is achieved using two integers, n and L, in the range of $[1, \mathrm{D}]$ which are generated

once per trial vector. n is randomly generated and is the donor vector's index from where the crossover with the trial vector begins. L is the actual amount of donor vector elements contributed to the trial vector starting from index n. L lies in the interval $[1, \mathrm{D}]$ and is generated using the pseudocode in Algorithm 2.

Algorithm 2 Exponential crossover.

```
n = rand(0, (D − 1))
L = 0
while (rand(0, 1) ≤ Cr) and (L ≤ D) do
    L = L + 1
end while
```

The exchange of elements after n and L are acquired is achieved as seen in Equation 2.6.

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{for } j \in \{n, \langle n+1 \rangle_D, \ldots, \langle n+L-1 \rangle_D\} \\ x_{j,i,G} & \text{for all other } j \in [1, \mathrm{D}] \end{cases} \tag{2.6}$$

The angular brackets, $\langle\rangle_D$, denote a modulo function with modulus D and are used to give the vectors a circular buffer effect. The n^{th} element of the trial vector is always populated with data from the n^{th} element of the donor vector, guaranteeing that the trial vector is at least made up from one donor vector element even if $L = 0$.

An example of the exponential crossover operation can be seen in Figure 2.4.

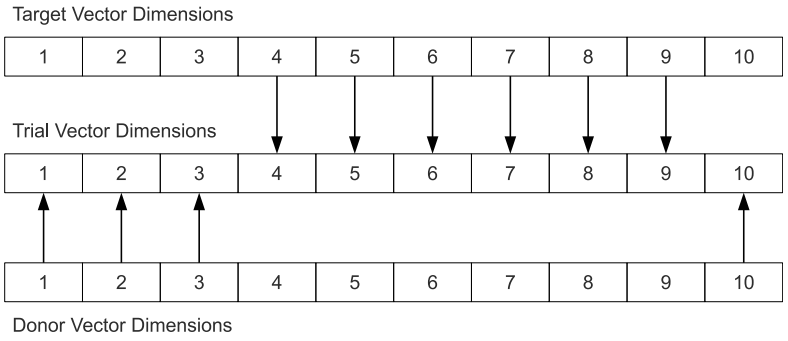


Figure 2.4: Elements exchanged sequentially using exponential crossover when $n = 10$ and $L = 4$.

2.3.2 Binomial Crossover

In binomial crossover, each j^{th} element of trial vector is populated arbitrarily with data from the j^{th} element of either the donor vector or the target vector. A uniformly distributed random number in the interval of $[0, 1]$ is generated for all D dimensions of all NP trial vectors. If the generated number is less than or equal to the crossover rate, Cr, the exchange of elements takes place—the i^{th} trial vector at index j is populated with data from the i^{th} donor vector at index j; otherwise it is populated with data from the j^{th} index of target vector. To guarantee that at least one element from the donor vector is copied to the trial vector, another random number, k_{rand}, in the range of $[1, \mathrm{D}]$ is generated per trial vector. Therefore, the j^{th} index of the i^{th} trial vector that equals k_{rand} will always be populated with data from the donor vector. Binomial crossover can be expressed as seen in Algorithm 3 and Equation 2.7.

Algorithm 3 Binomial crossover.

for $i = 1 \rightarrow NP$ **do**
  $k_{rand} = rand(1, D)$
  for $j = 1 \rightarrow D$ **do**
    if $(rand(0, 1) \leq Cr)$ or $(j = k_{rand})$ **then**
      $trialVector[i][j] = donorVector[i][j]$
    else
      $trialVector[i][j] = targetVector[i][j]$
    end if
  end for
end for

$$u_{j,i,G} = \begin{cases} v_{j,i,G} & \text{if } (\mathrm{rand}_{i,j}[0, 1] \leq Cr \text{ or } j = k_{rand}) \\ x_{j,i,G} & \text{otherwise} \end{cases} \tag{2.7}$$

An example of a typical run of the binomial crossover method can be observed in Figure 2.5.

2.3.3 Literature on Crossover

Ken Price claims that the crossover method is not particularly important and that the binomial method is never worse than exponential[9]. Jeyakumar and Shanmugavelayutham supported this claim stating that *"Regardless of the characteristics and dimen-*

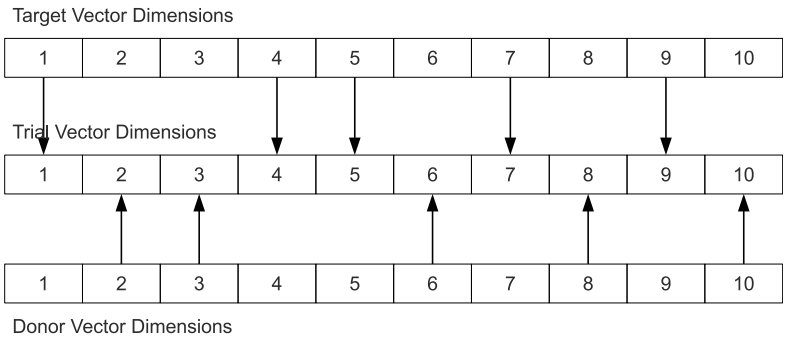


Figure 2.5: Elements exchanged arbitrarily using binomial crossover.

sion of the functions, relatively better results seem to have been provided by the variants with binomial crossover[6]*."*

Daniela Zaharie researched the effects of binomial and exponential crossover in DE and reported that the crossover probability of the binomial method is linear, whereas of exponential it is not[10].

According to Gämperle et al., a large Cr value can expedite convergence, but there is a limit to how large said value can be—although the tip point where the performance begins to deteriorate is different for every objective function, it usually lies in the range of $[0.9, 1]$[4].

2.4 Selection

The selection step is the equivalent to Darwinism's "survival of the fittest" paradigm. During the selection step, the quality of the target (i.e., the parent) and trial (i.e., the offspring) vectors is brought to direct confrontation. The vectors are tested against the objective function and the vector that returns the best result survives to the next generation, i.e., $G = G + 1$.

The equations that follow are for the minimization or maximization of an objective function respectively:

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G} & \text{if } f(\vec{U}_{i,G}) \leq f(\vec{X}_{i,G}) \\ \vec{X}_{i,G} & \text{if } f(\vec{U}_{i,G}) > f(\vec{X}_{i,G}) \end{cases} \tag{2.8a}$$

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G} & \text{if } f(\vec{U}_{i,G}) \geq f(\vec{X}_{i,G}) \\ \vec{X}_{i,G} & \text{if } f(\vec{U}_{i,G}) < f(\vec{X}_{i,G}) \end{cases} \tag{2.8b}$$

$f(\vec{X})$ is the objective function to be minimized or maximized. Ergo, if the trial vector provides a better or equal solution than the target vector, it replaces the target vector at $G = G + 1$; otherwise the target vector remains the same for the next generation.

2.5 Objective Function

Throughout the execution of the algorithm, DE attempts to minimize or maximize an objective function. An objective function takes a D-sized vector that is referred to as *candidate solution* as a parameter and uses that data to calculate and return the result of the function. A candidate solution represents a possible solution to the objective function.

An example code for an objective function for a 2-dimensional sinc function $\left(\frac{sin(x)}{x} \cdot \frac{sin(y)}{y}\right)$ can be seen in Algorithm 4.

Algorithm 4 Sinc objective function.

```
function ObjectiveFunction(candidateSolution[ ])
    result = 1
    for i = 1 → D do
        result = result · (sin(candidateSolution[i])/candidateSolution[i])
    end for
    return result
end function
```

2.5.1 Literature on Objective Function

According to the DE website, selecting the right objective function is essential. In the case of misconvergence, the choice of objective function should be considered because a function that better describes the problem could exist. Moreover, an objective function that incorporates all knowledge regarding the problem at hand can make a difference [9].

2.6 Objective Result

Using an objective result can save the algorithm a lot of processing time. The objective result can be any value as long as the value is not in contradiction with the purpose of the optimization—minimization or maximization.

Assigning an objective result of 0 while requesting the maximization of an objective function that returns values in the interval of $[0, 1]$ would likely cause the algorithm terminate prematurely and produce the wrong result. Provided that a correct objective result is given, the execution time of the algorithm could reduce dramatically because when the objective function returns a value that equals the objective result, the algorithm will terminate instead of unnecessarily evaluating the function for G_{max} iterations.

2.7 Terminating Conditions

The terminating condition signals the end of the DE algorithm and can be one of the following: (1) a predefined and fixed maximum number of iterations, G_{max}, with a large enough value to accommodate the complexity of the objective function; (2) a lack of change in the fitness of the population for a significant amount of iterations; and lastly (3) a pre-defined objective result has been reached by a member of the population.

Condition (1) is mostly used for measuring the performance of an algorithm—the less iterations an algorithm needed to solve, the better it performed. Additionally, conditions (1) and (2) can act as a fail-safe for preventing an infinite loop by terminating when the algorithm has not found a solution in G_{max} iterations, or by terminating because the algorithm is stuck at a local optimum and the fitness of the population has not changed significantly. When condition (3) has been activated, it means that an actual solution to the problem has been found. A combination of all three conditions can be used, but using condition (2) can be problematic as it might not allow enough time for the algorithm to escape a local optimum—what is considered "a lack of change" or "a significant amount of iterations" is too subjective.

2.8 Putting it All Together

In this section, the classic DE scheme "DE/rand/1/bin" is put together. "DE/rand/1/bin" mutates using a random element as a base vector and one difference for perturbation.

Producing offspring is achieved using the binomial recombination method where elements are exchanged arbitrarily. The pseudocode for the classic DE scheme can be seen in Algorithm 5.

Algorithm 5 Classic DE scheme: "DE/rand/1/bin"

Step 1: Read control parameters F, Cr and NP
Step 2: Set generation to zero: $Generation = 0$
Step 3: Perform initialization
while the ending criterion is not satisfied **do**
  for $i = 1 \rightarrow NP$ **do**
    Step 4.1: Perform mutation
    Step 4.2: Perform crossover
    Step 4.3: Perform selection
  end for
  Step 5: Increase the generation count: $G = G + 1$
  Step 6: Evaluate
end while

Example source code for each of the steps of the classic DE scheme is provided in Appendix B. Listing B.1 presents the way in which the population is initialized, Listing B.2 shows the implementation of the DE/rand/1 mutation scheme, Listing B.4 lists the source for the binomial crossover method and Listing B.5 shows the greedy selection step that is used for the minimization of an objective function. Lastly, Listing B.6 provides an example code for an objective function.

2.9 Variants

Since its inception in 1995, DE has attracted the interest of people from all scientific fields, leading to the development of a plethora of DE variants. Some of the variants were created for specific uses and applications; others were just seen as changes that could improve the algorithm as a whole. In this section. the most important DE variants are listed.

2.9.1 SaDE

Even though mutation strategies and crossover methods provide the user with options in the quest of tackling a problem, not every strategy is effective against all problems[11].

In an attempt to provide a solution to the problem of the DE scheme choice, Qin et al. proposed a method for enabling self-adaptation in DE, named SaDE[12]. SaDE contains a pool of four DE schemes: DE/rand/1/bin, DE/rand-to-best/2/bin, DE/rand/2/bin and DE/current-to-rand/1. Initially, all schemes have an has an equal chance of being selected to produce an offspring, but as the algorithm progresses, the variant keeps track of a "success and failure rate" and uses it to increase or decrease the probability of selection of the successful and unsuccessful schemes respectively.

2.9.2 DE/rand/1/either-or

Many times, the objective function that is considered for optimization cannot be solved using conventional settings such as $Cr = [0.1, 0.9]$, leading to a significant amount of trial-and-error being spent before the solution comes in the form of a pure mutant or pure recombinant vector.

Price et al. proposed a technique that produces either pure mutants, either pure recombinants depending on the value of a control parameter p_f. Even though the scheme is mainly aimed at functions that are better solved by pure mutants or pure recombinants, a comparative study has shown that the state-of-the-art technique produces competitive results compared to classic DE variants such as DE/rand/1/bin and DE/target-to-best/1[13].

2.9.3 Jitter/Dither

Since tuning the control parameters proved to be quite a challenge, Price et al. coined two new terms for the randomization of the scaling factor, F: *jitter* and *dither*[14].

Jitter refers to the generation of a new random F value for every j^{th} element of every i^{th} vector. In constrast, dither refers to the generation of a random F value for every i^{th} vector, but not for every element.

2.10 Drawbacks

Rönkkönen et al. have reported that DE yields a poor performance when dealing with hyprid composition functions[5].

The inadequecy of selective pressure during the mutation step becomes obvious when dealing with non-seperable problems as reported by Sutton et al[15].

2.11 Research Directions

Although DE is an extremely powerful real-valued optimizer, there are still many open problems for what are considered areas for improvement. Suganthan et al. dedicated a section to future research in their literature review[3].

The paper listed eight "open problems" or future research directions. Although all eight research ideas can potentially improve DE, problems 5 and 6 look particularly interesting for future research.

Problem 5 refers to a lack of selective pressure during the mutation stage which can lead to inefficient exploitation. Sutton et al. first referred to this, suggesting the incorporation of rank-based selection scheme during the mutation stage of the algorithm[15]. Implementing this change in the mutation step can be achieved by using a ranking table where the fittest parents have a higher chance of being drawn as indices. Balancing exploration and exploitation is a delicate procedure, however, as population diversity can easily be lost in the presence of excessive selective pressure. Consequently, a time-sensitive selection pressure technique was proposed. During the early stages of the algorithm, exploration is favored, so the selection pressure is almost non-existant, but as the algorithm progresses, the selection pressure increases, favoring exploitation.

Problem 6 discusses the greedy selection criterion implemented by DE where the parent and its offspring confront each other and the winner survives. The nature of the selection stage implies that DE is sensitive to the choice of initial population as this can lead to premature convergence[16]. According to Suganthan et al. *"A very interesting future research topic may be to integrate the mutation operator taken from DE with the non-elitist, fitness proportionate selection, without crossover that can result in good saddle crossing abilities of an EA."*

2.12 Summary

DE attempts to optimize a problem by searching for a global optimum while minimizing or maximizing an objective function in regards to the requirements of a problem. The basic DE algorithm consists of four steps: (1) initialization of the population in a predefined search space, (2) mutation, (3) crossover, and finally (4) selection. The initialization step takes place only at "Generation Zero." Mutation, crossover, and selection are all taking place per generation until a terminating condition has been satisfied.

The performance of DE is dictated by the three control parameters: the population size, NP; the scaling factor, F; and the crossover rate, Cr. While many papers have

been published regarding the optimal values for the control parameters, no combination exists that provides the best performance for all types of problems. Additionally, the abundance of DE schemes along with their inability to produce consistent results for all types of problems, led to the development of a number of DE variants that automate the process of tuning the control parameters and choosing the best mutation scheme and crossover method possible.

Bibliography

[1] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] R. Storn and K. Price, "Differential evolution: A simple evolutionary strategy for fast optimization," *Dr. Dobb's Journal*, vol. 22, no. 4, pp. 18–24, 1997.

[3] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[4] R. Gämperle, S. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Advanced in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS Press, Interlaken, Switzerland*, pp. 293–298, 2002.

[5] J. Rönkkönen, S. Kukkonen, and K. Price, "Real-parameter optimization with differential evolution," *IEEE Congress on Evolutionary Computation*, pp. 506–513, 2005.

[6] G. Jeyakumar and C. Shanmugavelayutham, "Convergence analysis of differential evolution variants on unconstrained global optimization functions," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 2, no. 2, 2011.

[7] R. Storn, "On the usage of differential evolution for function optimization," *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), IEEE, Berkeley*, pp. 519–523, 1996.

[8] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *ICSI, USA*, 1995.

[9] F. Ahlers, W. D. Carlo, C. Fleiner, L. Godwin, M. Keenan, R. Nath, A. Neumaier, J. Phillips, K. Price, R. Storn, P. Turney, F.-S. Wang, J. V. Zandt, H. Geldon,

P. Gauden, C. Brauer, K. Shivaram, and D. Novikov, "Differential evolution website." http://www.icsi.berkeley.edu/~storn/code.html. [Online; last accessed on February 20 2012].

[10] D. Zaharie, "A comparative analysis of crossover variants in differential evolution," *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 171–181, 2007.

[11] V. Feoktistov and S. Janaqi, "Generalization of the strategies in differential evolution," *Proc. 18th IPDPS*, p. 165a, 2004.

[12] A. Qin, V. Huang, and P. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, 2009.

[13] S. Das, A. Abraham, U. Chakraborty, and A. Konar, "Differential evolution using a neighborhood based mutation operator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 526–553, 2009.

[14] K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2005.

[15] A. Sutton, M. Lunacek, and L. Whitley, "Differential evolution and non-separability: Using selective pressure to focus search," *Proc. 9th Annu. Conf. GECCO*, pp. 1428–1435, 2007.

[16] R. Storn, *Advances in Differential Evolution*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2008.

Chapter 3

CEC 2005 Test Suite

Since the inception of Evolutionary Computation, many mathematical optimization algorithms have been introduced. Initially, measuring the performance of the algorithms was simply a matter of attempting to solve any set of problems to prove that the technique worked. Due to the lack of standard tests, the algorithms were tested against different sets of problems, or even the problems tested happened to be the same, they differed in dimensionality, constraints etc. This inconsistency in the testing methodology brought results that offered minimal or no insight on how well an algorithm performed against its peers. The inexistence of a common testing methodology, in addition to the increasing amount of algorithms introduced, made the comparison of the performance of different algorithms extremely difficult. What was missing was standardization.

Suganthan et al. published a paper titled "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization"[1] in an attempt to standardize the method for measuring the performance of optimization algorithms. The paper listed a test suite of twenty-five problems along with a performance measuring scheme for empirically measuring the performance of optimization algorithms.

In this part of the report, the *Differential Evolution Toolkit* (DET) is tested against five of the CEC 2005 Test Suite problems.

3.1 Testbed

The CEC Test Suite included twenty-five problems that were categorized into different properties. The main properties a problem can have are the following:

Shifted

This means that a function's global optimum has been shifted by a D amount of offsets, with D referring to the dimensions of a problem. For instance, if a 2 dimensional objective function whose global optimum is at $(0, 0)$ is shifted by $(-10, 10)$, its global optimum has been shifted and is now located at coordinates $(10, -10)$.

Scalable

Refers to the ability of a function to adapt (or scale) to a user's increased demands.

A scalable function does not have a fixed amount of dimensions or search space. For example, 2-dimensional function in the search space of $[-100, 100]$ is not scalable, unless the user is able to modify the number of dimensions and alter the search space without having to re-write the function.

Unimodal/Multimodal

Refers to the amount of modes (or optima) a function can have. For instance, for a unimodal function, f, in the interval of $[a, b]$ and with x^* as its global optimum, the function's returned value from a to x^* increases and from x^* to b decreases — therefore, the function has only one optimum which is global. This, however, is not the case for multimodal functions where there could be more than one local or even global optima.

Separable/Non-separable

Refers to the ability of a function, $F(x, y)$, to be separated into single variable functions, $f(x)$, $f(y)$. Then, by providing the appropriate mathematical symbol between the two independent functions, the result of said equation equals, $F(x, y)$. A function that satisfies these criteria is called separable; otherwise it is non-separable.

Noisy

Noise is used to arbitrarily alter the result during each evalution, making the result impossible to predict even when the candidate solution and offset values are known. Noisy functions are difficult to solve because of their ability to mislead the algorithm.

Choosing five out of the twenty-five CEC 2005 problems was a matter of choosing problems that would provide the best insight on how well the DET performed. Therefore, the problems chosen had different properties as this was seen as an opportunity to observe the effects of the population size, NP, scaling factor, F, and crossover rate, Cr, on the convergence rate of DE.

The five problems that were chosen from the CEC 2005 Suite[1] to test the DET were the following:

1. F_1: Shifted Sphere Function (Unimodal/Separable)
2. F_2: Shifted Schwefel's Problem (Unimodal/Non-separable)
3. F_4: Shifted Schwefel's Problem With Noise in Fitness (Unimodal/Non-separable)

4. F_6: Shifted Rosenbrock's Function (Multimodal/Non-separable)

5. F_9: Shifted Rastrigin's Function (Multimodal/Separable)

3.2 Testing

3.2.1 Testing Methodology

F was initialized to 0.1 and Cr to 0.0. Both control parameters were given a step size of $+0.1$ and a maximum value of 1.0. All problems have a dimensionality of $D = 10$, and NP was set to $(10 \cdot D + 1 = 101)$[1]as this was seen as a good compromise between convergence probability and function evaluations[2]. The search space was set in the range of $[-1000.0, 1000.0]$. Following the findings of Jeyakumar and Shanmugavelayutham[3], the algorithm was set to use the classic DE method of "DE/rand/1/bin." The maximum amount of generations, G_{max} was fixed to a moderate 10000, so as to test the efficacy of the algorithm in solving problems.

Using a $+0.1$ step size for both F and Cr resulted in 11 possible F values ($[0.1, 1.0]$) and 10 possible Cr values ($[0.0, 1.0]$), which when multiplied results in the amount runs needed to cover all combinations of the two control parameters. Therefore, each problem was run a total of 110 times (10×11). Running each combination just once, however, is not enough to provide sound data. Only if the results can be replicated are the results considered valid. Therefore, each run was set to to execute 100 times. In total, every problem was run 110 times and every run executed 100 times. The results for each run were the mean of all 100 executions.

Regarding the objective-function-specific details: the objective of the DET is to minimize the functions. All problems have a global optimum of 0.0, but a tolerance error value was given, hence the objective result was fixed to 1×10^{-12}. The algorithm only terminates before the maximum amount of generations, G_{max}, has been reached if the objective function returns a result that is less than or equal to the objective result. It is important to note that the population and objective function offsets were re-initialized per execution. A list of the execution details per problem is seen in Table 3.1.

[1]The population size was set to 101 instead of 100 to allow for the direct comparison of the results with the *DE/best-to-next/1* mutation scheme which is discussed in Chapter 4.

	F_1	F_2	F_4	F_6	F_9
D:	10	10	10	10	10
NP:	101	101	101	101	101
Generations:	10000	10000	10000	10000	10000
Mutation:	rand/1	rand/1	rand/1	rand/1	rand/1
Crossover:	bin	bin	bin	bin	bin
Runs:	110	110	110	110	110
Executions:	100	100	100	100	100
Global Optimum:	1×10^{-12}	1×10^{-12}	1×10^{-12}	1×10^{-12}	1×10^{-12}
Search Space:	$[-1000.0, 1000.0]$	$[-1000.0, 1000.0]$	$[-1000.0, 1000.0]$	$[-1000.0, 1000.0]$	$[-1000.0, 1000.0]$

Table 3.1: The execution details for the five CEC problems.

3.2.2 Empirical Measurements

In order to empirically measure the performance of DE, two aspects were considered: the speed at which the algorithm converged to a solution and the probability of the algorithm actually converging to a solution.

The first step of measuring the performance was to calculate the mean generations, G_m, needed for all successful executions. To calculate the mean generations, a sum of all generations passed during successful executions, G_t, was taken and divided by the number of successful executions, E_s, as seen in Equation 3.1.

$$G_m = \frac{G_t}{E_s} \tag{3.1}$$

Once the mean generations, G_m, has been calculated, the main quality measure, Q_m, is acquired by dividing the convergence probability, P_c, with the mean generations, G_m, as seen in Equation 3.2.

$$Q_m = \frac{P_c}{G_m} \tag{3.2}$$

where P_c is essentially the number of successful executions, E_s, multiplied by 100 and divided by the total number of executions requested.

Additionally, the quality measure, Q_m, is normalised by dividing every quality measure with the best quality measure (the one with the highest value) of each problem.

Extra measuring techniques were used in the form of calculating the minimum, maximum, average and error rate values per dimension, and the overall error rate for all members of the population.

Measuring the error rate is a technique used to calculate the "distance" of the algorithm from the expected outcome, i.e., the solution. The error rate per dimension can be calculated by taking the absolute value of the subtraction between the j^{th} offset and the mean of all members of the population at the j^{th} dimension as seen below.

$$error[j] = |offset[j] - average[j]| \tag{3.3}$$

After all error rates per dimension have been acquired, the overall error rate is the summation of all error rates divided by the number of dimensions, D.

In the next pages, there can be found a group of two graphs, and a table with the ten best performing combinations of F and Cr. The first graph in each group shows a 3-D representation of the quality measure, Q_m, each set took to solve; and the second in each group shows the overall error rate of the best performing set of F and Cr for each problem. Minimum-maximum-average and error graphs per dimension for all five problems are provided in Appendix A. Finally, an attempt is made to provide conclusive data for the proper tuning of the DE control parameters.

3.3 Problem F1

$$F_1(x) = \sum_{i=1}^{D} z_i^2 \tag{3.4}$$

where D: dimensions, $z = x - o$, $x = [x_1, x_2, \ldots, x_D]$ are the elements of the candidate solution and $o = [o_1, o_2, \ldots, o_D]$ is the shifted global optimum.

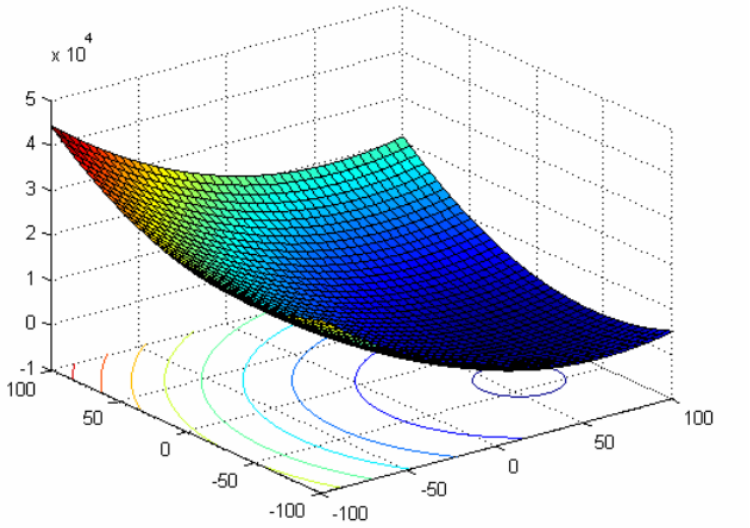


Figure 3.1: Shifted Sphere Function - taken from: CEC 2005 Test Suite[1].

Properties

- Unimodal
- Shifted
- Separable
- Scalable

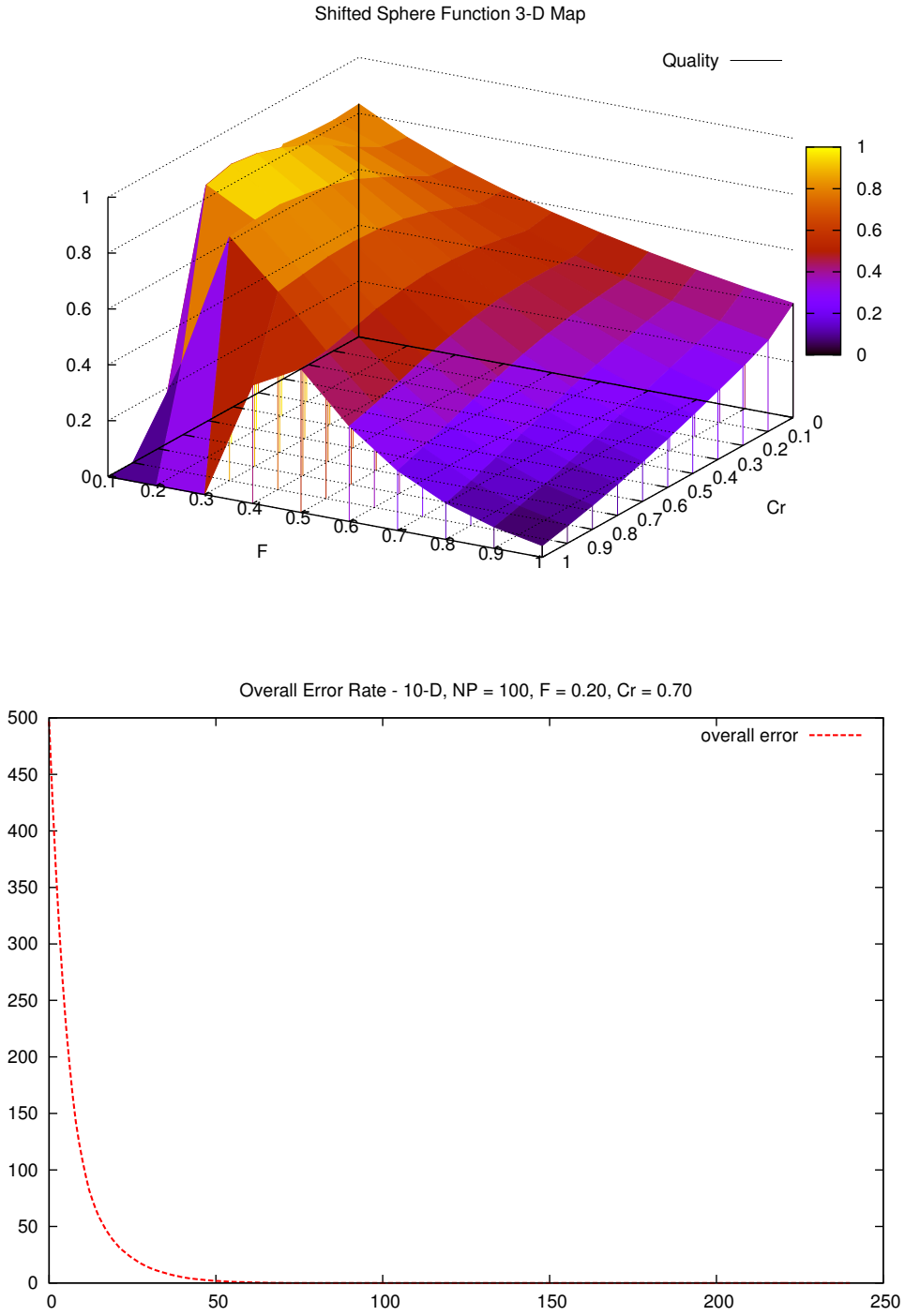


Figure 3.2: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Sphere Function.

Rank	F	Cr	G_m	P_c	Q_m
1	0.2	0.7	241.53	100	1.0000
2	0.2	0.6	244.63	100	0.9873
3	0.2	0.8	245.10	99	0.9756
4	0.2	0.5	251.80	100	0.9592
5	0.2	0.4	262.79	100	0.9191
6	0.3	0.7	271.80	100	0.8886
7	0.3	0.8	273.65	100	0.8826
8	0.3	0.6	275.97	100	0.8752
9	0.3	0.9	276.98	100	0.8720
10	0.2	0.3	278.50	100	0.8673

Table 3.2: Top 10 performing sets of F and Cr for the Shifted Sphere Function.

3.4 Problem F2

$$F_2(x) = \sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2 \tag{3.5}$$

where D: dimensions, $z = x - o$, $x = [x_1, x_2, \ldots, x_D]$ are the elements of the candidate solution and $o = [o_1, o_2, \ldots, o_D]$ is the shifted global optimum.

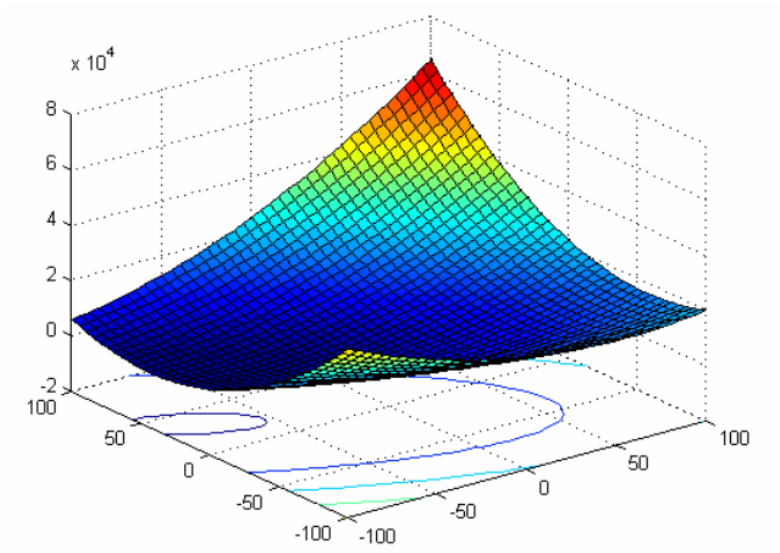


Figure 3.3: Shifted Schwefel's Problem - taken from: CEC 2005 Test Suite[1].

Properties

- Unimodal
- Shifted
- Non-separable
- Scalable

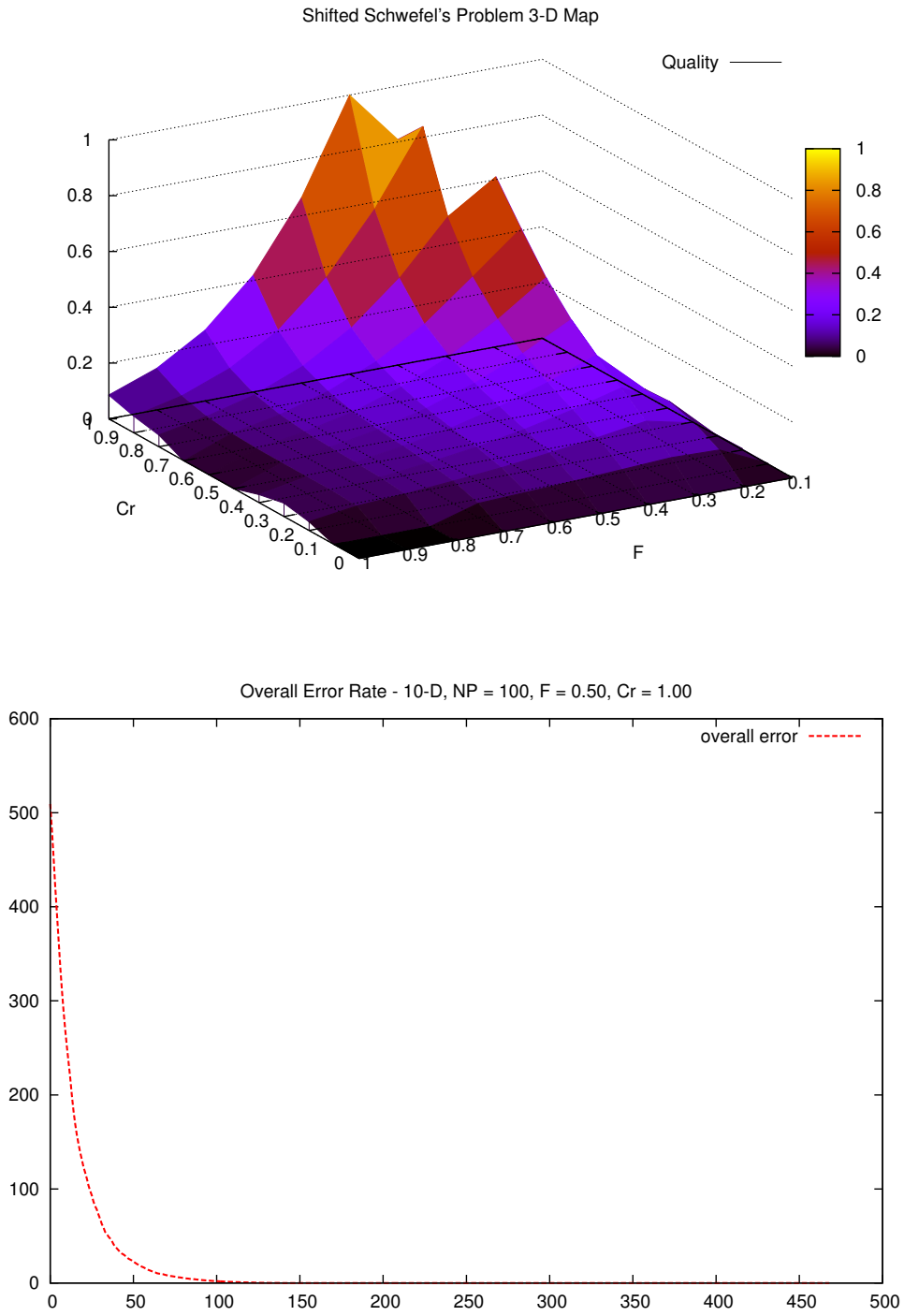


Figure 3.4: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Schwefel's Problem.

Rank	F	Cr	G_m	P_c	Q_m
1	0.5	1.0	470.95	100	1.0000
2	0.4	0.9	520.35	100	0.9051
3	0.4	1.0	349.53	60	0.8084
4	0.3	0.8	576.65	91	0.7432
5	0.6	1.0	709.55	100	0.6637
6	0.5	0.9	734.76	100	0.6410
7	0.4	0.8	745.52	100	0.6317
8	0.3	0.7	758.84	98	0.6082
9	0.3	0.6	977.12	100	0.4820
10	0.4	0.7	1009.20	100	0.4667

Table 3.3: Top 10 performing sets of F and Cr for the Shifted Schwefel's Problem.

3.5 Problem F4

$$F_4(x) = (\sum_{i=1}^{D}(\sum_{j=1}^{i} z_j)^2) \cdot (1 + 0.4 \cdot |N(0,1)|) \tag{3.6}$$

where D: dimensions, $z = x - o$, $x = [x_1, x_2, \ldots, x_D]$ are the elements of the candidate solution, $o = [o_1, o_2, \ldots, o_D]$ is the shifted global optimum and $N(0,1)$ is a random number in the range of $[0, 1]$.

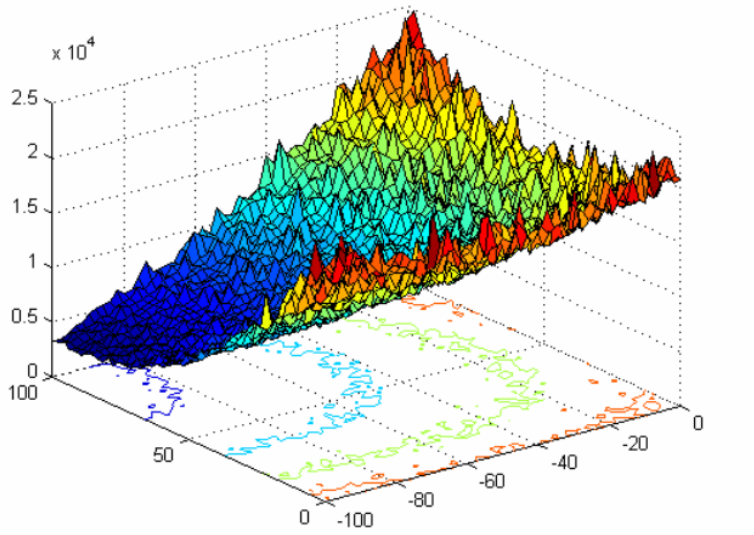


Figure 3.5: Shifted Schwefel's Problem With Noise in Fitness - taken from: CEC 2005 Test Suite[1].

Properties

- Unimodal
- Shifted
- Non-separable
- Scalable
- Noise in Fitness

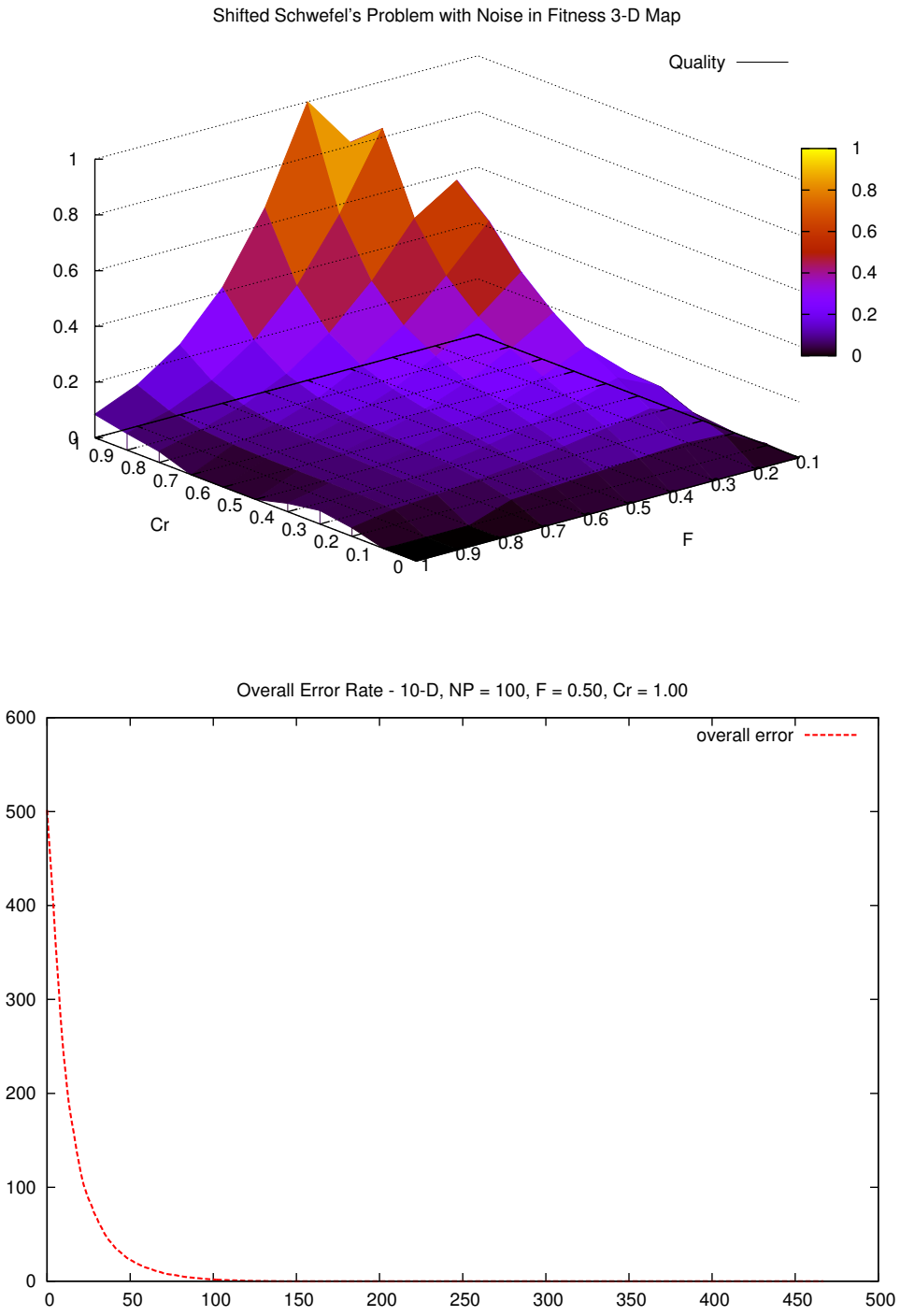


Figure 3.6: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Schwefel's Problem with Noise in Fitness.

Rank	F	Cr	G_m	P_c	Q_m
1	0.5	1.0	470.16	100	1.0000
2	0.4	0.9	518.52	100	0.9067
3	0.4	1.0	347.12	60	0.8127
4	0.3	0.8	571.48	88	0.7240
5	0.6	1.0	711.48	100	0.6608
6	0.5	0.9	734.84	100	0.6398
7	0.4	0.8	748.98	100	0.6277
8	0.3	0.7	756.98	100	0.6211
9	0.3	0.6	973.81	100	0.4828
10	0.4	0.7	1009.51	100	0.4657

Table 3.4: Top 10 performing sets of F and Cr for the Shifted Schwefel's Problem with Noise in Fitness.

3.6 Problem F6

$$F_6(x) = \sum_{i=1}^{D-1} (100 \cdot (z_i^2 - z_{i+1})^2 + (z_i - 1)^2) \tag{3.7}$$

where D: dimensions, $z = x - o + 1$, $x = [x_1, x_2, \ldots, x_D]$ are the elements of the candidate solution and $o = [o_1, o_2, \ldots, o_D]$ is the shifted global optimum.

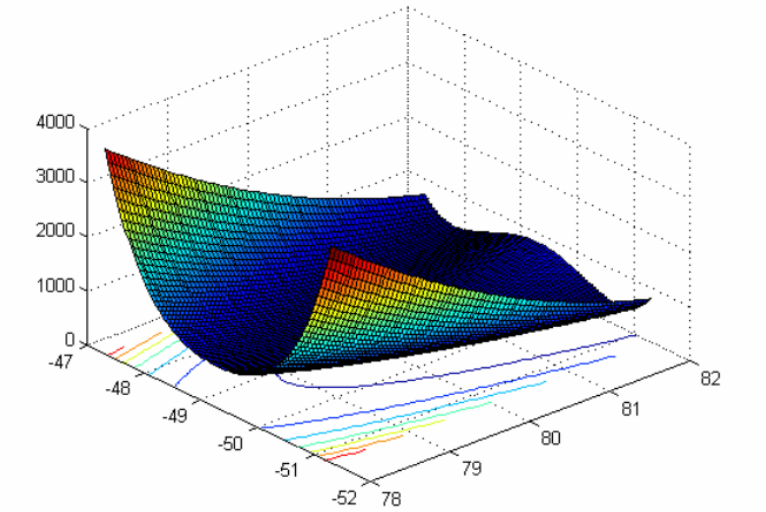


Figure 3.7: Shifted Rosenbrock's Function - taken from: CEC 2005 Test Suite[1].

Properties

- Multi-modal
- Shifted
- Non-separable
- Scalable
- Very Narrow Valley from Local Optimum to Global Optimum

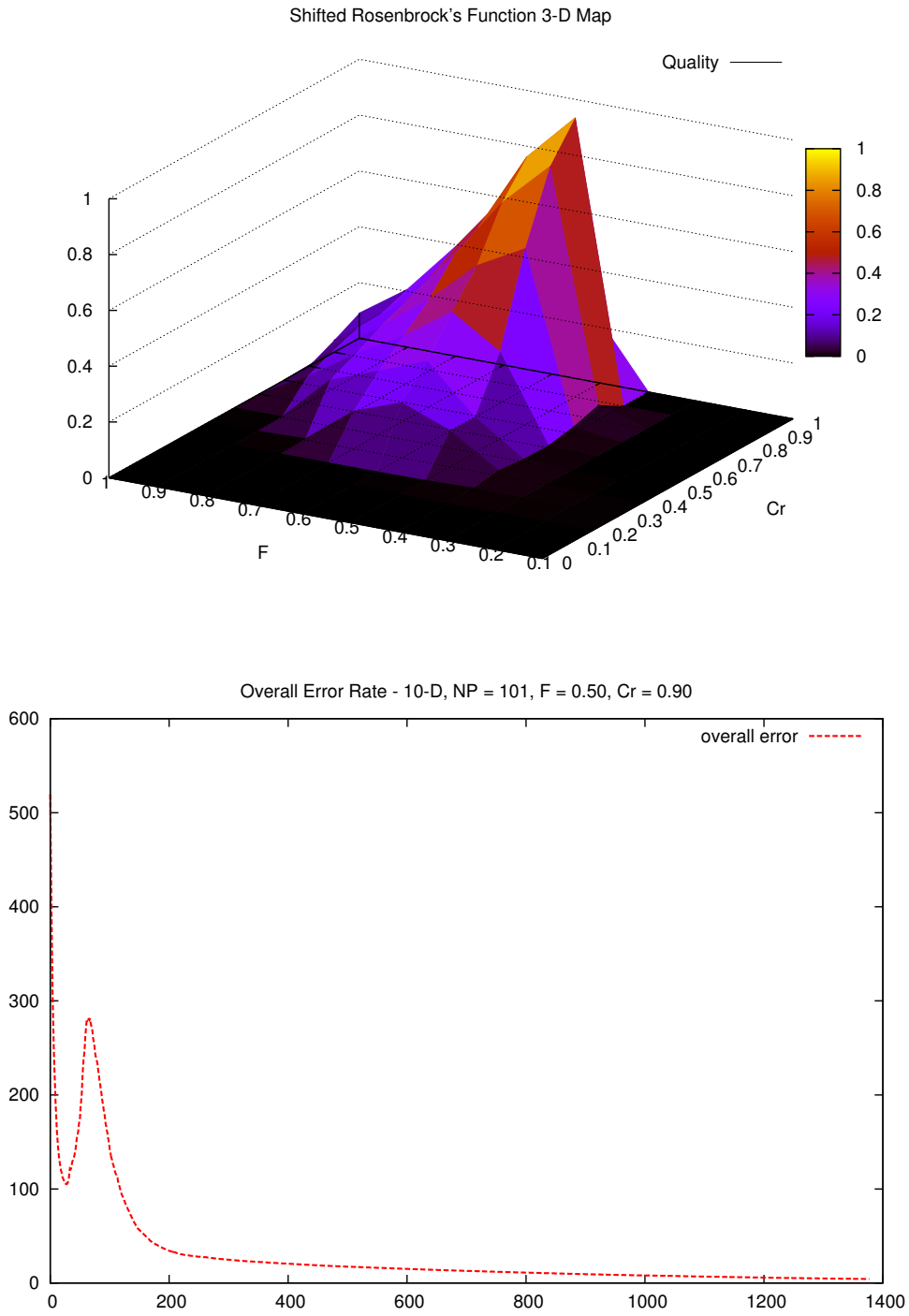


Figure 3.8: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Rosenbrock's Function.

Rank	F	Cr	G_m	P_c	Q_m
1	0.5	0.9	1428.06	95	1.0000
2	0.5	0.8	1664.47	97	0.8760
3	0.6	0.9	1742.90	96	0.8280
4	0.6	1.0	1480.40	78	0.7920
5	0.6	0.8	2084.54	99	0.7139
6	0.5	0.7	2381.68	100	0.6312
7	0.7	1.0	2329.53	93	0.6001
8	0.7	0.9	2676.89	97	0.5447
9	0.6	0.7	2788.37	100	0.5391
10	0.7	0.8	3062.70	100	0.4908

Table 3.5: Top 10 performing sets of F and Cr for the Shifted Rosenbrock's Function.

3.7 Problem F9

$$F_9(x) = \sum_{i=1}^{D}(z_i^2 - 10 \cdot cos(2\pi z_i) + 10) \tag{3.8}$$

where D: dimensions, $z = x - o$, $x = [x_1, x_2, \ldots, x_D]$ are the elements of the candidate solution and $o = [o_1, o_2, \ldots, o_D]$ is the shifted global optimum.

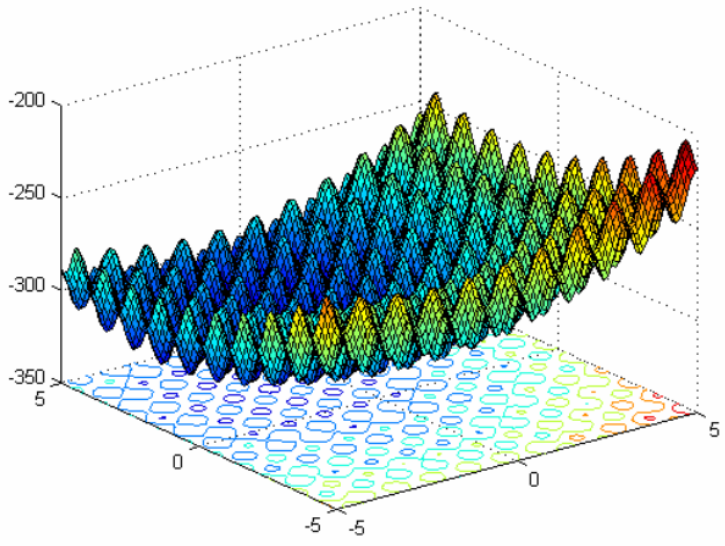


Figure 3.9: Shifted Rastrigin's Function - taken from: CEC 2005 Test Suite[1].

Properties

- Multi-modal
- Shifted
- Separable
- Scalable
- Huge Number of Local Optima

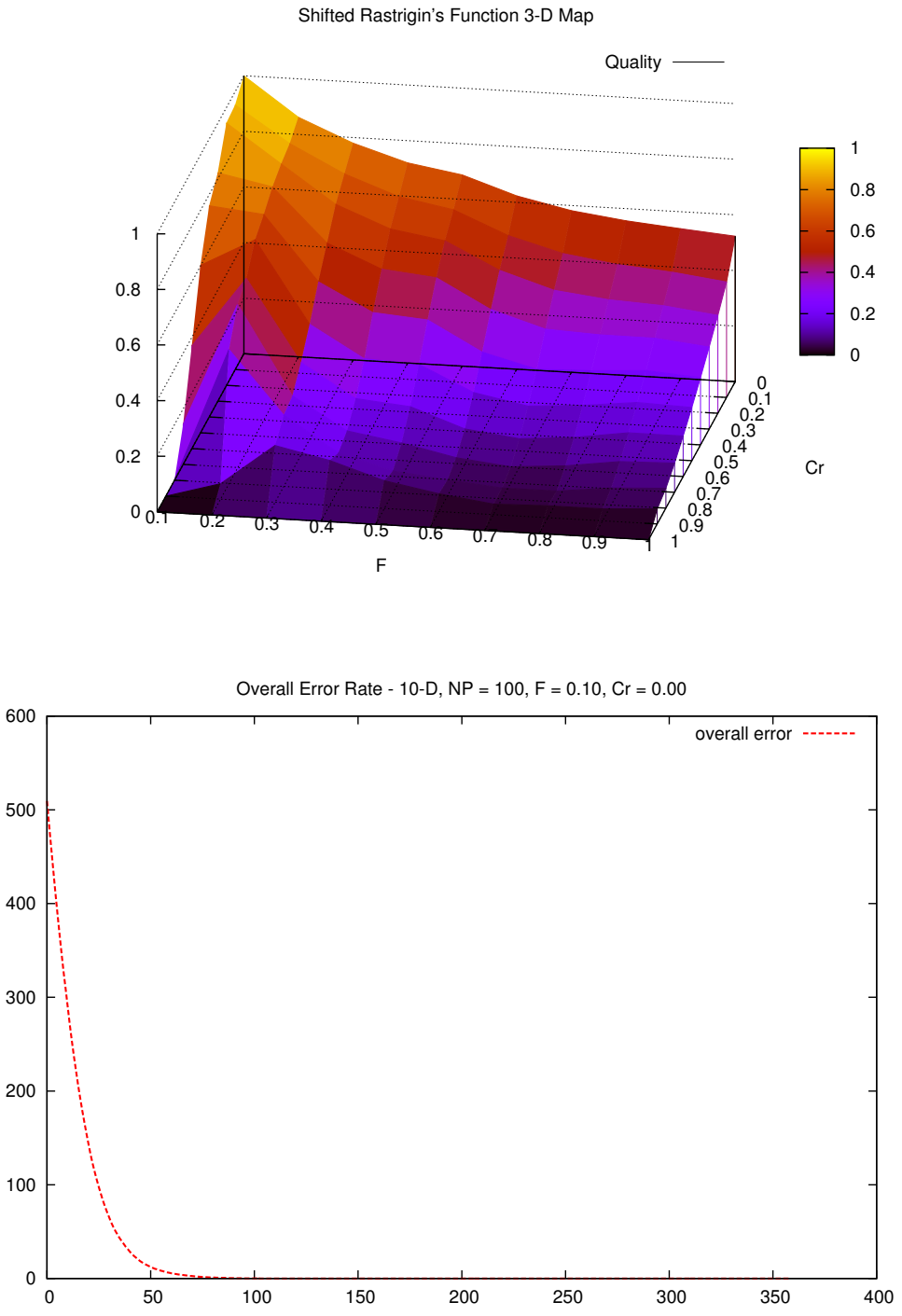


Figure 3.10: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Rastrigin's Function.

Rank	F	Cr	G_m	P_c	Q_m
1	0.1	0.0	358.79	100	1.0000
2	0.1	0.1	361.81	96	0.9520
3	0.1	0.2	353.44	93	0.9441
4	0.2	0.0	416.24	100	0.8620
5	0.2	0.1	435.37	100	0.8241
6	0.1	0.3	340.62	78	0.8216
7	0.2	0.2	443.99	100	0.8081
8	0.2	0.3	458.73	100	0.7821
9	0.3	0.0	460.00	100	0.7800
10	0.1	0.4	330.39	70	0.7601

Table 3.6: Top 10 performing sets of F and Cr for the Shifted Rastrigin's Function.

3.8 Results

Graphs and tables provide an insight to how the convergence rate of DE is affected by the different values of F and Cr. Since NP was fixed throughout all of the testing, the terms "combination" and "set" are used to denote the values of both F and Cr, unless explicitly stated.

Even though no combination of F and Cr made it into the top ten performing combinations of all problems, the common characteristics should help in the identification of good starting values for the scaling factor and crossover rate, thus reducing the required amount of trial-and-error needed to achieve the best performance possible. It is important to note that the following results were rounded to the closest 0.1 step size that had been used in the initial testing.

3.8.1 Running Times

To get a feel of the efficacy of the DET, its performance in a real-world context was measured. The relevant specifications of the computer that was used during testing are provided, along with data regarding the running time during the best and worst execution of each problem.

- Description: Notebook
- Model: HP Pavilion g6
- CPU: Intel Core i5-2410M
  - Frequency: 2.90GHz
  - Cores: 2
  - Threads: 4
  - Cache: 3MB
  - Width: 64 bits
- Memory: 4096MB
  - Frequency: 1333MHz
  - Width: 64 bits
- OS: Linux Mint 11 64-bit

Table 4.6 lists the running times for the best and worst combinations of F and Cr for each problem per single execution. The results are in milliseconds (ms) and in generations per millisecond (G/ms).

	F_1	F_2	F_4	F_6	F_9
Best (ms):	67	74	80	140	93
Best (G/ms):	3.61	6.43	7.74	21.27	3.70
Worst (ms):	1091	1367	1373	1133	2047
Worst (G/ms):	9.17	7.32	7.28	8.83	4.89

Table 3.7: Running time details for the five problems.

3.8.2 Unimodal

The combination listed below is the seventh top performing set in both the F_2 and F_4 problems, and also managed to solve F_1 with a success rate of 100% in a mean of 335.17 generations, scoring a quality measure of 0.7206. Moreover, an F value of 0.40 appears in $8/30$, and a Cr value of 0.80 appears in $6/30$ top sets, making this set a very attractive option when dealing with unimodal problems.

Mean F:	0.40
Mean Cr:	0.80

Table 3.8: Unimodal performance summary.

3.8.3 Multimodal

The combination listed in Table 3.9 is not particularly helpful when dealing with multimodal problems because the two multimodal problems are polar opposites — F_6 was extremely difficult to solve with a mere $30/110$ combinations of scaling factor and crossover rate managing to score a convergence probability of over 50%, whereas F_9 proved very easy to solve by almost all sets (this set included). That being said, $F = 0.40$ and $Cr = 0.50$ are only $+0.1$ shy from values that performed well against F_6, with $F = 0.50$ showing up thrice in the top ten combinations and $Cr = 0.60$ thrice in the top twenty combinations.

Mean F:	0.40
Mean Cr:	0.50

Table 3.9: Multimodal performance summary.

3.8.4 Seperable

The set listed in Table 3.10 proved to be a very attractive option for attempting to solve seperable problems. Even though it did not earn a place in the top combinations tables of both problems, it ranked fifth for the F_1 problem and just missed out on the top ten table for the F_9, being ranked eleventh. More importantly, $F = 0.20$ appears in $10/20$ and $Cr = 0.40$ in $2/20$ top sets, making this a very good starting point when a seperable problem is at hand.

Mean F:	0.20
Mean Cr:	0.40

Table 3.10: Seperable performance summary.

3.8.5 Non-seperable

The set listed below proved to be very powerful for dealing with non-seperable problems. To begin with, it ranked second in the extremely difficult to solve F_6 problem, and ranked eleventh for F_2 and F_4 problems. Additionally, an F value of 0.50 and a Cr value of 0.80 both appear seven times in the top thirty combinations.

Mean F:	0.50
Mean Cr:	0.80

Table 3.11: Non-seperable performance summary.

3.8.6 Overall Findings

Even though the combination listed in Table 3.12 does not appear in all of the top ten tables, looking into the detailed data that were collected by the DET, the combination would make an excellent starting point for solving all five problems. More specifically, it ranked tenth for problems F_2 and F_4, while also solving problems F_1 and F_9 100% successfully. However, it performed badly against problem F_6 scoring a convergence probability of 2%. Nonetheless, looking at the top combinations table of problem F_6, it

can be observed that $Cr = 0.70$ appears in two out of the top ten combinations for this problem, and an increase of the scaling factor to 0.50 yields the sixth placed combination for problem F_6, making this an excellent starting point for attempting to solve any kind of problem.

Mean F:	0.40
Mean Cr:	0.70

Table 3.12: Overall performance summary.

Due to the fact that this would be an excellent starting point for solving any kind of problems, allowing for an offset of ± 0.1 results in more interesting data. The combinations that can be derived by using the offset are seen in Table 3.13. The leftmost and rightmost elements refer to the scaling factor and crossover rate respectively.

0.30/0.60	*0.30/0.70*	*0.30/0.80*
0.40/0.60	**0.40/0.70**	*0.40/0.80*
0.50/0.60	*0.50/0.70*	*0.50/0.80*

Table 3.13: Combinations derived with an offset of ± 0.10 applied to $F = 0.40$ and $Cr = 0.70$.

The sets listed appear in an impressive $15/50$ top combinations, with the exception being the F_9 problem. All nine sets did manage, however, to perform extremely well against the F_9 problem, scoring a mean quality measure, Q_m, of 0.1949 with a convergence probability of 99%. More impressively, the $F = 0.50, Cr = 0.80$ and $F = 0.50, Cr = 0.70$ sets ranked second and sixth respectively for the very difficult F_6 problem.

3.9 Conclusion

The ability of the DET to solve these five problems has truly showcased the power of DE when optimizing real-valued functions. The problems that were put against the DET had different properties, were high-dimensional, and the Shifted Rosenbrock's Function was a particularly tough one to solve, but the DET has managed to solve them, and in most cases doing so with a significant amount of generations to spare. The fact that these problems had different properties further proves the power of DE against all types of optimization problems.

The most impressive feat was the almost seamless transition between solving the two Shifted Schwefel's problems: F_2 and its F_4 counterpart which included noise in

fitness. Solving the noisy problem had minimal or no impact at all on the mean generations the algorithm needed to solve. In fact, when comparing the top combinations tables for both problems, the top combinations tables are identical, with the difference in quality measure, Q_m, being marginal at ± 0.01.

That being said, however, getting the DET to work at the highest capacity requires some trial-and-error on the user's behalf until the ideal combination of values for the three control parameters is found. Judging from the data gathered during this experiment though, a large enough population size in combination with the use of the sets listed in this section (particularly $F = 0.40$ and $Cr = 0.70$, in addition to all sets listed in Table 3.13) as a rule-of-thumb should at least provide a good enough starting point for any problem at hand.

Bibliography

[1] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," 2005.

[2] J. Rönkkönen, S. Kukkonen, and K. Price, "Real-parameter optimization with differential evolution," *IEEE Congress on Evolutionary Computation*, pp. 506–513, 2005.

[3] G. Jeyakumar and C. Shanmugavelayutham, "Convergence analysis of differential evolution variants on unconstrained global optimization functions," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 2, no. 2, 2011.

Chapter 4

The "best-to-next" Mutation Scheme

One of the biggest issues with biologically-inspired optimization algorithms is the loss of "genetic" diversity, which causes the algorithm to converge prematurely to a local optimum. Within Differential Evolution, the crossover stage is responsible for maintaining the diversity of the population by populating the trial vector with data from the target and donor vectors. The population does not always contain enough "genetic" diversity and the crossover stage alone is not flexible enough to prevent misconvergence. DE crossover relies on the donor vector which is produced during the previous stage: mutation. Hence, we may presume that "genetic" diversity within the population is heavily dependent upon the mutation stage.

By analogy, the best chefs in the world would not produce miracles if they were given rotten tomatoes, and the effectiveness of the crossover stage is reliant on the diversity provided by the mutation operator—which provides no means of measuring the quality of the donor vector.

Many mutation schemes have been offered in the literature, but their implementation is stochastic in nature which brings about difficulties in predicting the quality of the resulting donor vector. For that reason, in an attempt to provide an alternative to the stochastic mutation techniques, the *DE/best-to-next/1* mutation scheme has been developed. The intention was to create a non-stochastic method for mutating the parent in order to guarantee a balanced population quality, whilst, simultaneously, improving the speed of convergence and the probability of escaping local optima.

In this part of the report, the *DE/best-to-next/1* mutation scheme is introduced and tested against the same set of five problems from the CEC Test Suite[1] that was used to test the *Differential Evolution Toolkit* (DET) in Chapter 3.

4.1 Hypothesis

In Differential Evolution, four stages exist: initialization, mutation, crossover and selection. Initialization occurs once, whereas mutation, crossover and selection are put together in a cycle of operations that takes place at every generation until the algorithm has terminated.

Although this is a cycle of operations, a better-fitting name would be an eccentric cycle of operations. By definition, all operations in a cycle are assigned the same weight

and there is no actual beginning. However, that is not the case for DE. Selection cannot function without a trial vector produced by the crossover stage; crossover cannot function without a donor vector produced by the mutation stage; mutation only needs the target vector which is produced by the initialization stage that is not part of the cycle. Thus, it is imperative to guarantee that the donor vector produced during the mutation stage is populated with quality data that can be used by the following stages. Assigning a quality measure to the donor vector is not an easy task, however. What is it that makes a quality donor vector, after all?

4.1.1 Donor Vector Quality

At the current stage, state-of-the-art Differential Evolution variants stochastically mutate the target vector to produce a donor vector[2]. Owing to the stochasticity of the mutation schemes, predicting the outcome of the mutation process is impossible. Therefore, a technique which produces the donor vector in a non-stochastic manner needs to be considered, thus ensuring a consistent donor vector quality across all DE generations. Before a non-stochastic mutation scheme is developed, it must first be decided what makes a quality donor vector. To achieve that, we need to delve deeper into DE's mechanics, and consequently to nature, which is the inspiration for most evolutionary algorithms, including DE.

In the real world, one could argue that the population's quality is balanced—there exist the "best," the "average" and the "worst" members of the population. Intuitively, it would seem that robust children can only be produced when elite members of the population reproduce. However, people from different categories (ranking-wise) mingle and produce offspring, and the outcome is not always predictable. For example, two strong parents can produce weak children, while two weak parents can produce strong children.

With this in mind, it became apparent that the real-world coupling was essentially meant to be simulated during the crossover stage—two distinct members of the population exchanging genes to produce offspring. However, a more careful analysis reveals that this is not exactly the case. During the crossover stage, the target and donor vectors are recombined to produce the trial vector, which in a real-world context would be analogous to two parents producing an offspring. In the DE case, though, there exists a paradox. The target and donor vectors do not represent the male and the female during reproduction because unlike the real-world coupling of parents, they are not two distinct members of the population; the donor vector is, in essence, a mutant target vector. Because of that, the impact of "crossing genes over" is diminished when the target

and donor vectors are similar, a phenomenon which will likely occur because the target vector is stochastically mutated. Consequently, the mutation stage itself should be responsible for guaranteeing that the resulting donor vector is dissimilar to the target vector at every generation.

4.2 Implementation and Testing

4.2.1 Implementation – A Deterministic Mutation Scheme

During the mutation stage, the target vector is perturbed using a vector as a base and one or more scaled difference vectors. Instead of stochastically mutating the target vector and producing an unknown-quality donor vector, an attempt was made to provide a deterministic mutation scheme which would guarantee the production of a diverse donor vector that is dissimilar to the target vector. This was achieved by modifying the *DE/best/1* mutation scheme to enable *DE/best-to-next/1* to fully exploit the scaled difference vector aspect of the mutation stage. The modification resulted in a deterministic mutation scheme that uses the best agent of every generation as its base and a scaled difference vector that perturbs using both very different and very similar vectors, thus producing a balanced donor vector.

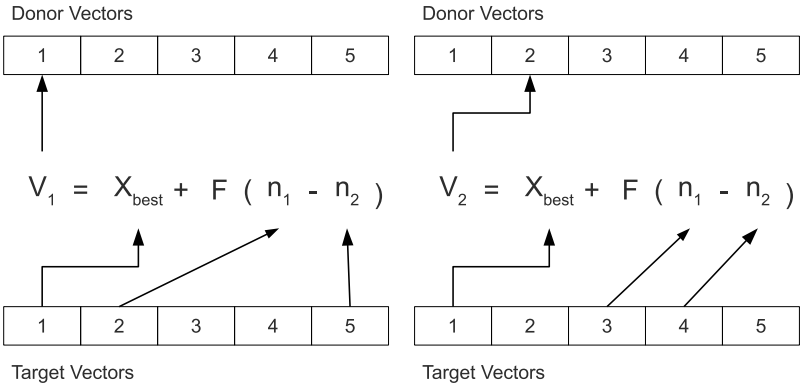
To achieve this balance, an odd-numbered population size, NP, is required. After each generation's, G, evaluation stage, the population is sorted by fitness with the best vector at index zero.

Finding which two elements will make up the scaled difference is achieved by having two variables, n_1^i and n_2^i, initially point to the first element of the target vector at index one, and the last element at index $(NP-1)$ respectively. n_1^i will increment and n_2^i will decrement with each member of the population, thus pointing to the next best and next worse population members until n_1^i points to where n_2^i was initially pointing and vice-versa. The best member will be copied to the last index of the donor vector as is, forcing it to recombine with the worst member of the target vector. This is represented in Equation 4.1.

$$\text{“DE/best-to-next/1”: } \vec{V}_{i,G} = \vec{X}_{best,G} + F \cdot (\vec{X}_{n_1^i,G} - \vec{X}_{n_2^i,G}) \tag{4.1}$$

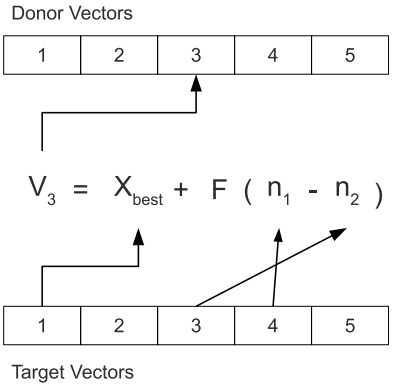
An example algorithm for the *best-to-next* mutation operation can also be seen in Algorithm 6. Source code for the scheme can also be found in Appendix B, Listing B.3.

In addition, Figure 4.2.1 shows a step-by-step diagram of the *best-to-next/1* mutation for a population size of $NP = 5$.

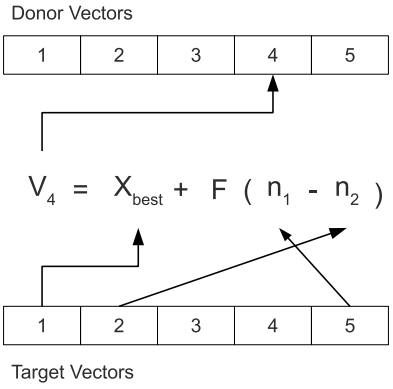


(a) Step 1

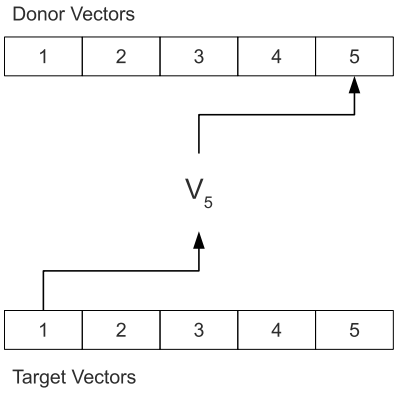
(b) Step 2



(c) Step 3



(d) Step 4



(e) Step 5

Figure 4.1: Step-by-step representation of the *best-to-next/1* mutation scheme for $NP = 5$.

Algorithm 6 best-to-next Mutation Scheme.

for $i = 0 \rightarrow (NP - 1)$ **do**
  $n_1 = i + 1$
  $n_2 = NP - n_1$
  for $j = 0 \rightarrow D$ **do**
    $donorVector[i][j] = targetVector[best][j] + F * (targetVector[n1][j] - targetVector[n2][j])$
  end for
end for
for $j = 0 \rightarrow D$ **do**
  $donorVector[NP - 1][j] = targetVector[best][j]$
end for

4.2.2 Testing Methodology

The conditions under which the *DE/best-to-next/1* mutation scheme was tested were identical to the ones listed in Section 3.2.1. The *DE/best-to-next/1* mutation scheme was used instead of *DE/rand/1*. All other details were kept exactly the same to allow for the comparison of the two set of results. Additionally, all performance measurements were made as described in Section 3.2.2.

Three new variables were introduced to allow the comparison of the two DE variants. Δ_{Q_m} refers to the difference in quality measure; Δ_F refers to the difference in the scaling factor control parameter; and Δ_{Cr} refers to the difference in the crossover rate control parameter.

4.3 Results

4.3.1 Problem F1

Rank	DE/best-to-next/1/bin					DE/rand/1/bin					Δ_{Q_m}
	F	Cr	G_m	P_c	Q_m	F	Cr	G_m	P_c	Q_m	
1	0.4	0.7	78.97	100	1.0000	0.2	0.7	241.53	100	0.3270	$+0.6730$
2	0.5	0.9	85.44	100	0.9243	0.2	0.6	244.63	100	0.3228	$+0.6015$
3	0.4	0.6	91.23	100	0.8656	0.2	0.8	245.09	99	0.3190	$+0.5466$
4	0.5	0.8	99.29	100	0.7953	0.2	0.5	251.80	100	0.3136	$+0.4817$
5	0.3	0.4	101.83	100	0.7755	0.2	0.4	262.79	100	0.3005	$+0.4750$
6	0.3	0.5	97.18	92	0.7476	0.3	0.7	271.80	100	0.2905	$+0.4571$
7	0.4	0.5	106.54	100	0.7412	0.3	0.8	273.65	100	0.2886	$+0.4526$
8	0.5	0.7	112.48	100	0.7021	0.3	0.6	275.97	100	0.2862	$+0.4159$
9	0.3	0.3	122.04	100	0.6471	0.3	0.9	276.98	100	0.2851	$+0.3620$
10	0.4	0.6	125.08	100	0.6314	0.2	0.3	278.50	100	0.2836	$+0.3478$

Table 4.1: Top 10 performing sets of F and Cr for the Shifted Sphere Function.

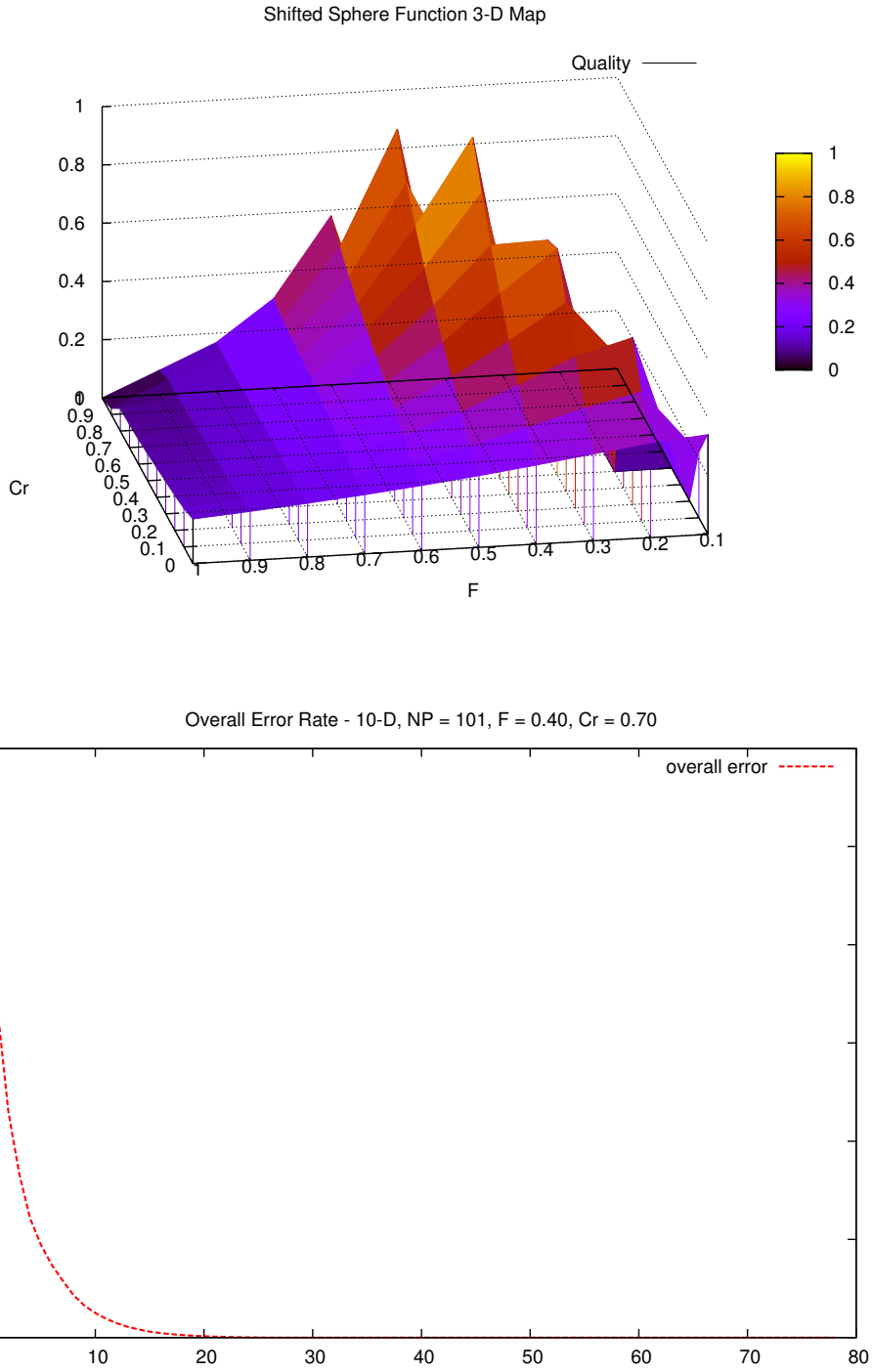


Figure 4.2: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Sphere Function.

4.3.2 Problem F2

Rank	DE/best-to-next/1/bin					DE/rand/1/bin					Δ_{Q_m}
	F	Cr	G_m	P_c	Q_m	F	Cr	G_m	P_c	Q_m	
1	0.5	0.9	132.09	91	1.0000	0.5	1.0	470.95	100	0.3082	$+0.6918$
2	0.6	1.0	133.62	91	0.9886	0.4	0.9	520.35	100	0.2789	$+0.7097$
3	0.4	0.7	173.34	100	0.8374	0.4	1.0	349.53	60	0.2492	$+0.5882$
4	0.5	0.8	186.21	100	0.7795	0.3	0.8	576.65	91	0.2291	$+0.5504$
5	0.6	0.9	220.26	100	0.6590	0.6	1.0	709.55	100	0.2046	$+0.4544$
6	0.4	0.6	234.83	100	0.6181	0.5	0.9	734.76	100	0.1975	$+0.4206$
7	0.3	0.5	245.49	100	0.5913	0.4	0.8	745.52	100	0.1947	$+0.3966$
8	0.7	1.0	258.98	100	0.5605	0.3	0.7	758.84	98	0.1875	$+0.3730$
9	0.5	0.7	267.87	100	0.5419	0.3	0.6	977.12	100	0.1486	$+0.3933$
10	0.4	0.5	328.54	100	0.4418	0.4	0.7	1009.20	100	0.1438	$+0.2980$

Table 4.2: Top 10 performing sets of F and Cr for the Shifted Schwefel's Problem.

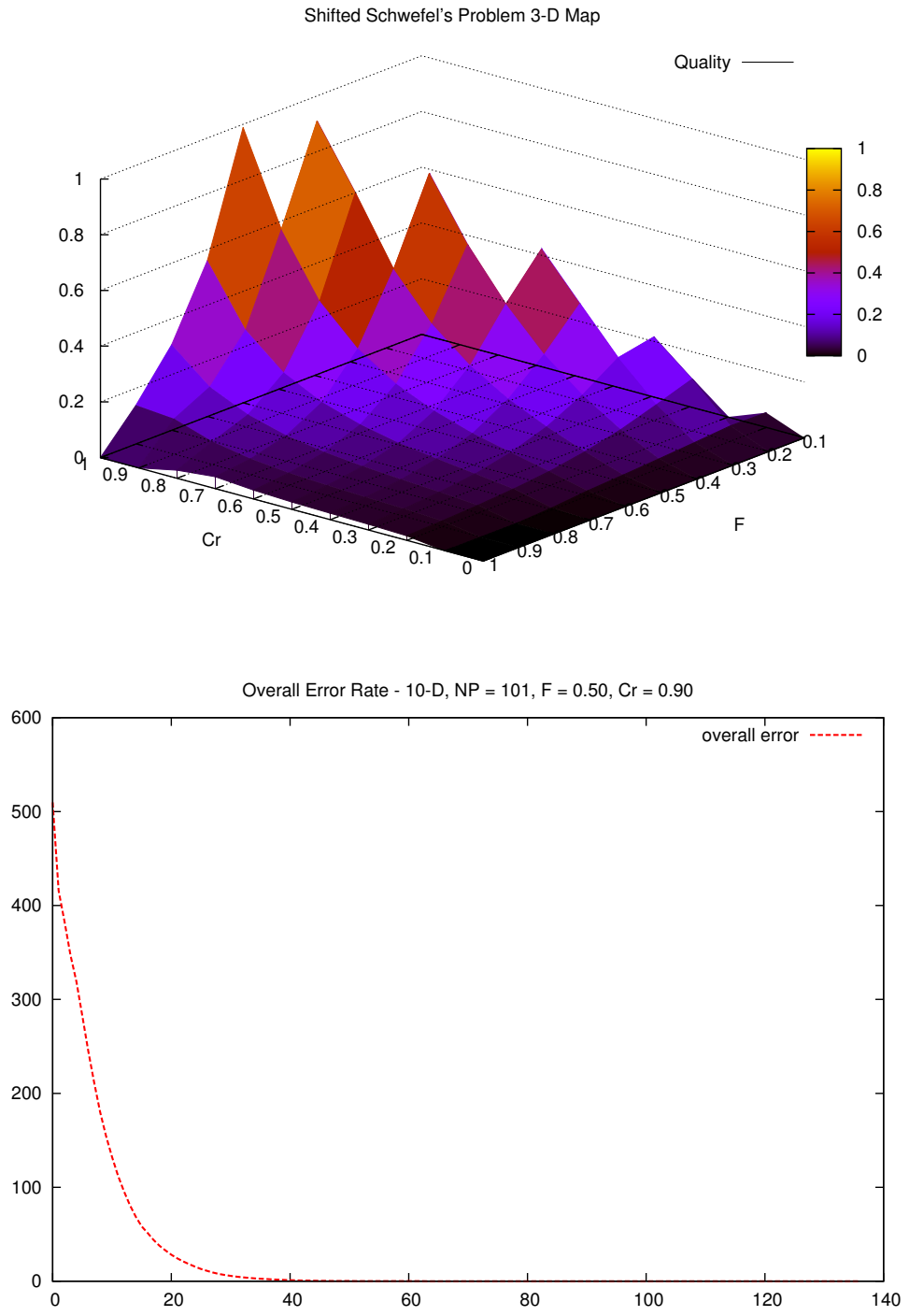


Figure 4.3: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Schwefel's Problem.

4.3.3 Problem F4

Rank	DE/best-to-next/1/bin					DE/rand/1/bin					Δ_{Q_m}
	F	Cr	G_m	P_c	Q_m	F	Cr	G_m	P_c	Q_m	
1	0.6	1.0	133.32	90	1.0000	0.5	1.0	470.16	100	0.3151	$+0.6849$
2	0.5	0.9	130.70	88	0.9974	0.4	0.9	518.52	100	0.2857	$+0.7117$
3	0.4	0.7	180.30	99	0.8134	0.4	1.0	347.12	60	0.2561	$+0.5573$
4	0.5	0.8	185.14	100	0.8001	0.3	0.8	571.48	88	0.2281	$+0.5720$
5	0.6	0.9	218.14	100	0.6769	0.6	1.0	711.48	100	0.2082	$+0.4707$
6	0.4	0.6	235.42	100	0.6292	0.5	0.9	734.84	100	0.2016	$+0.4276$
7	0.7	1.0	259.52	100	0.5708	0.4	0.8	748.98	100	0.1978	$+0.3730$
8	0.3	0.5	253.42	95	0.5553	0.3	0.7	756.98	100	0.1957	$+0.3596$
9	0.5	0.7	268.75	100	0.5512	0.3	0.6	973.81	100	0.1521	$+0.3991$
10	0.4	0.5	329.63	100	0.4494	0.4	0.7	1009.51	100	0.1467	$+0.3027$

Table 4.3: Top 10 performing sets of F and Cr for the Shifted Schwefel's Problem with Noise in Fitness.

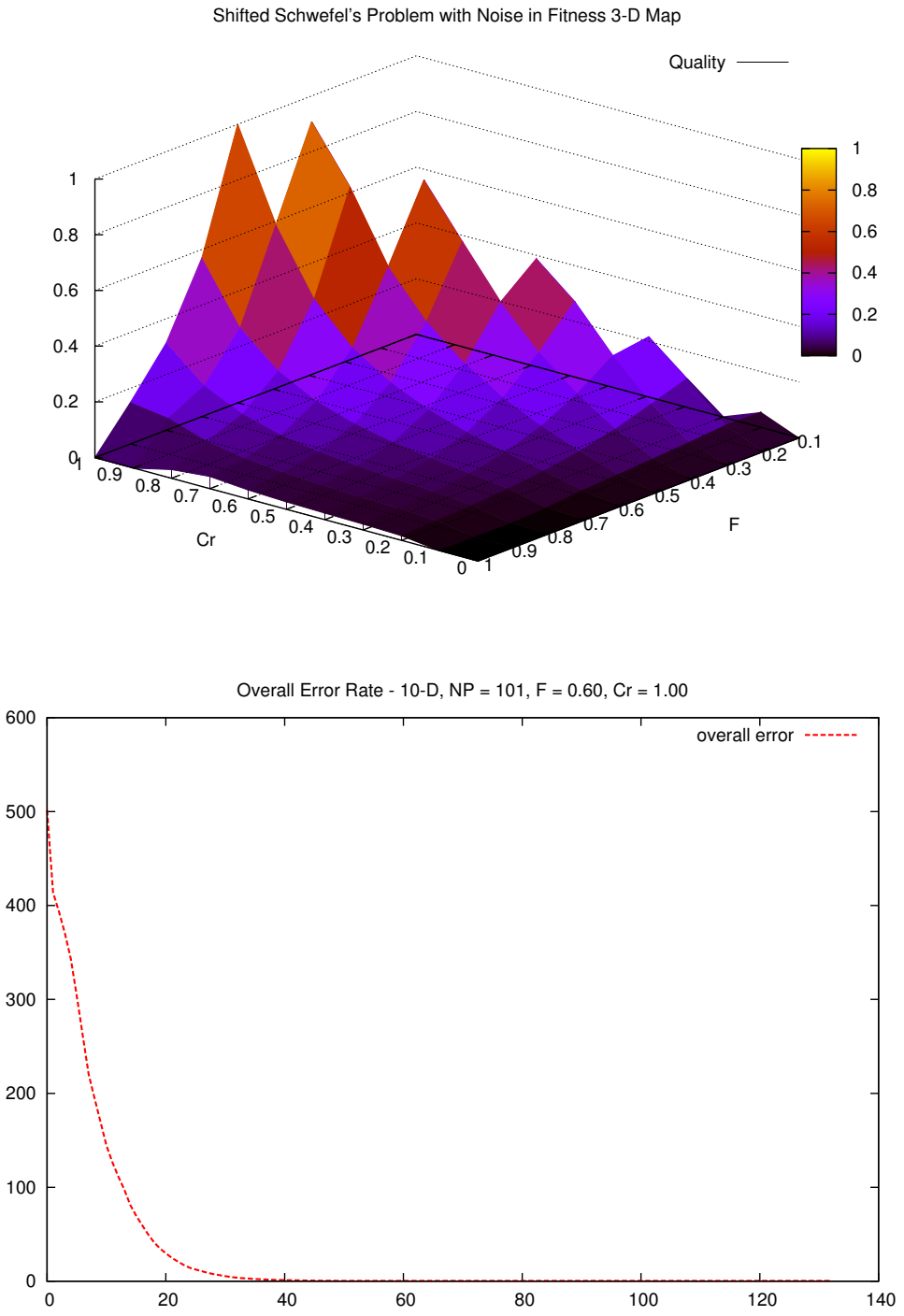


Figure 4.4: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Schwefel's Problem with Noise in Fitness.

4.3.4 Problem F6

Rank	DE/best-to-next/1/bin					DE/rand/1/bin					Δ_{Q_m}
	F	Cr	G_m	P_c	Q_m	F	Cr	G_m	P_c	Q_m	
1	0.6	0.9	403.45	75	1.0000	0.5	0.9	1428.06	95	0.3579	$+0.6421$
2	0.6	0.8	500.94	83	0.8913	0.5	0.8	1664.47	97	0.3135	$+0.5778$
3	0.7	0.9	575.35	88	0.8228	0.6	0.9	1742.90	96	0.2963	$+0.5265$
4	0.5	0.7	545.79	82	0.8082	0.6	1.0	1480.40	78	0.2834	$+0.5248$
5	0.7	1.0	409.81	52	0.6826	0.6	0.8	2084.54	99	0.2555	$+0.4271$
6	0.6	0.7	754.60	92	0.6558	0.5	0.7	2381.68	100	0.2259	$+0.4299$
7	0.7	0.8	839.18	91	0.5833	0.7	1.0	2329.53	93	0.2148	$+0.3685$
8	0.8	1.0	788.17	83	0.5665	0.7	0.9	2676.89	97	0.1949	$+0.3716$
9	0.5	0.6	859.46	78	0.4882	0.6	0.7	2788.37	100	0.1929	$+0.2953$
10	0.8	0.9	1071.75	79	0.3965	0.7	0.8	3062.70	100	0.1756	$+0.2209$

Table 4.4: Top 10 performing sets of F and Cr for the Shifted Rosenbrock's Function.

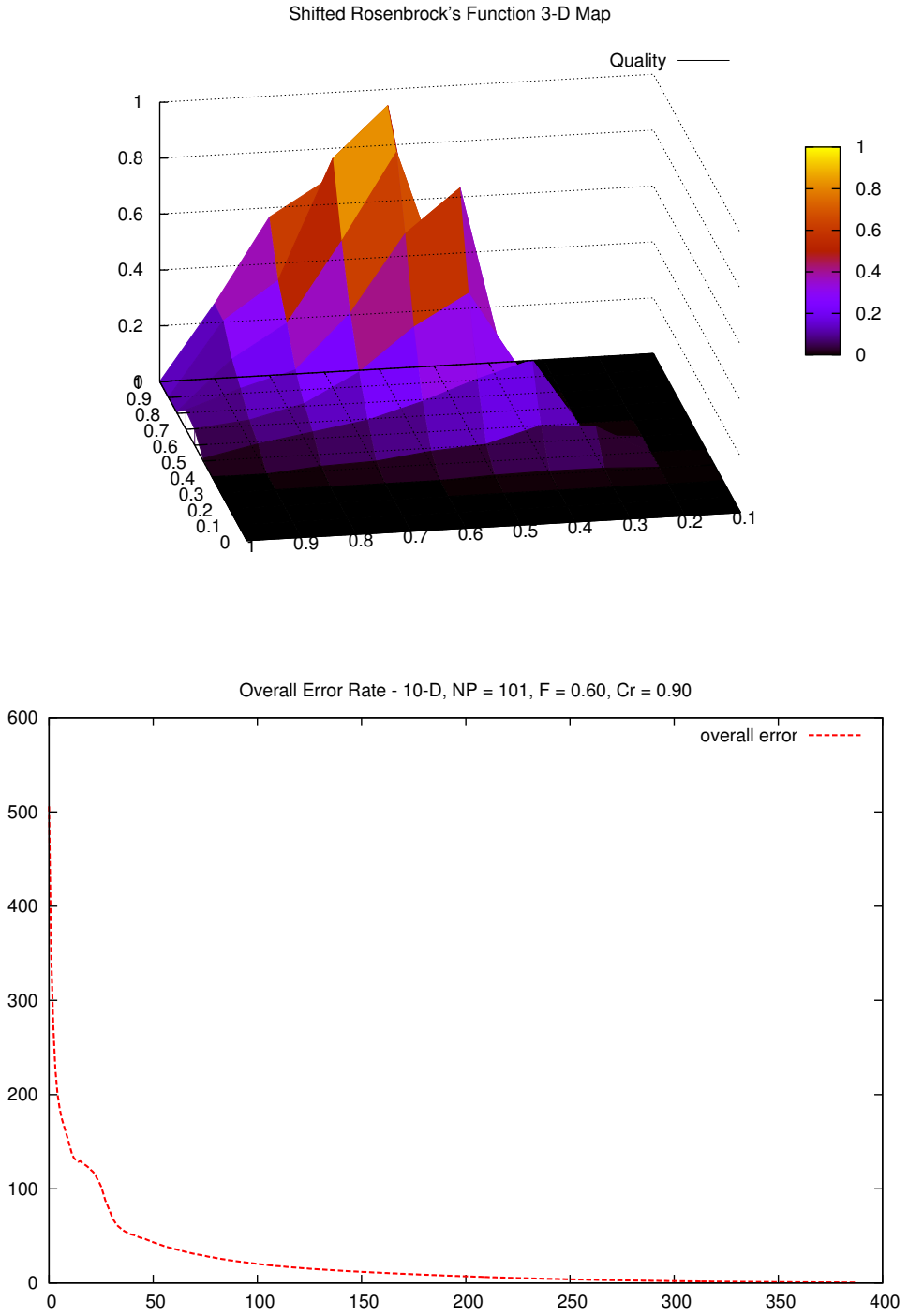


Figure 4.5: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Rosenbrock's Function.

4.3.5 Problem F9

Rank	DE/best-to-next/1/bin					DE/rand/1/bin					Δ_{Q_m}
	F	Cr	G_m	P_c	Q_m	F	Cr	G_m	P_c	Q_m	
1	0.2	0.0	318.52	96	1.0000	0.1	0.0	358.79	100	0.9248	$+0.0752$
2	0.3	0.0	339.92	100	0.9761	0.1	0.1	361.81	96	0.8803	$+0.0958$
3	0.5	0.2	326.60	95	0.9651	0.1	0.2	353.44	93	0.8730	$+0.0924$
4	0.5	0.1	356.56	100	0.9305	0.2	0.0	416.24	100	0.7971	$+0.1334$
5	0.4	0.0	371.55	100	0.8930	0.2	0.1	435.37	100	0.7621	$+0.1309$
6	0.4	0.1	327.05	85	0.8623	0.1	0.3	340.62	78	0.7598	$+0.1025$
7	0.5	0.0	390.95	100	0.8487	0.2	0.2	443.99	100	0.7473	$+0.1014$
8	0.5	0.3	304.65	75	0.8168	0.2	0.3	458.73	100	0.7233	$+0.0935$
9	0.6	0.1	429.60	99	0.7646	0.3	0.0	460.00	100	0.7213	$+0.0433$
10	0.6	0.0	447.96	100	0.7407	0.1	0.4	330.39	70	0.7030	$+0.0377$

Table 4.5: Top 10 performing sets of F and Cr for the Shifted Rastrigin's Function.

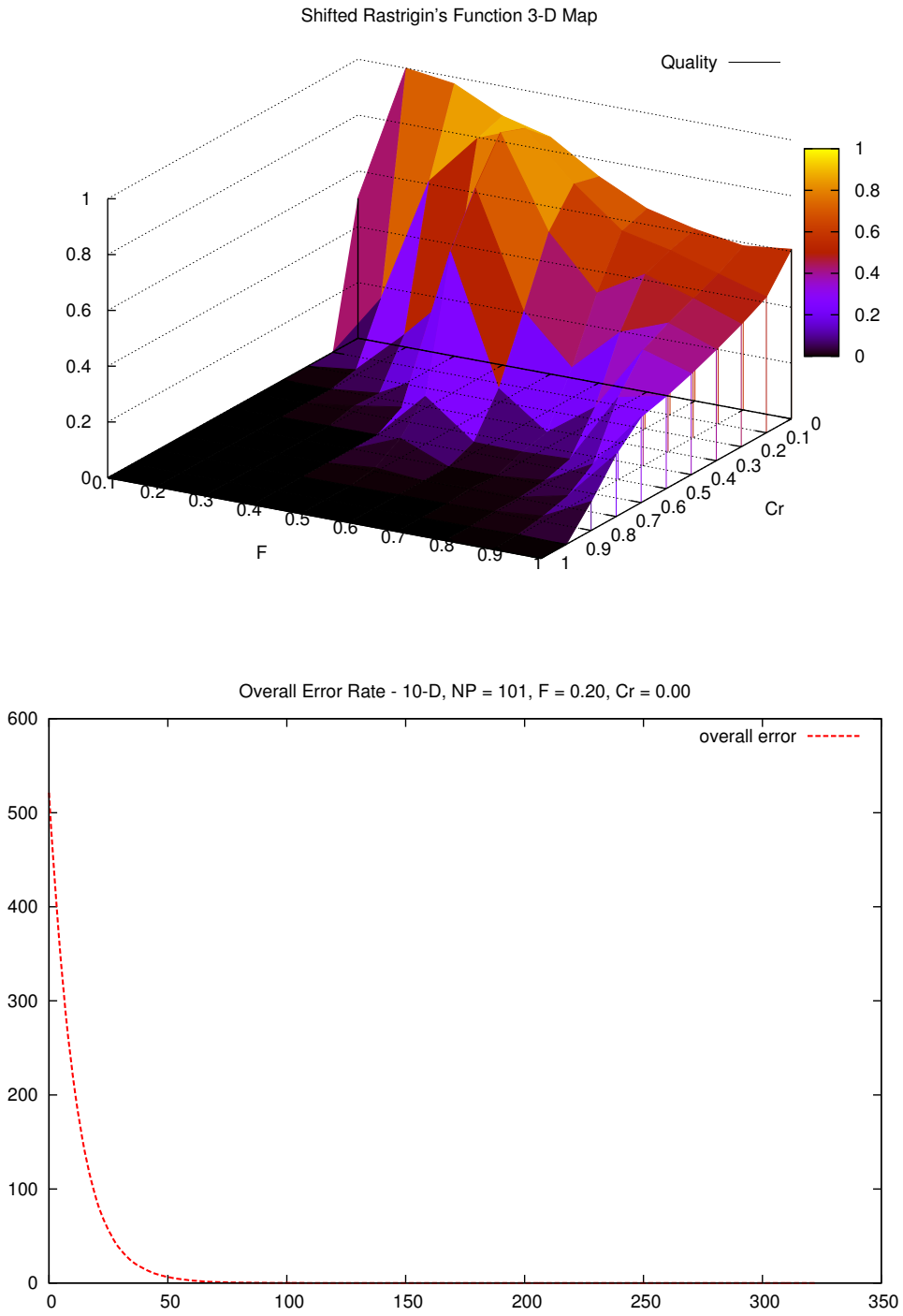


Figure 4.6: (a) The effect of the scaling factor, F, and the crossover rate, Cr, on the quality measure, Q_m, and (b) the overall error for the best performing set of F and Cr for the Shifted Rastrigin's Function.

Results

4.3.6 Running Times

Table 4.6 includes the execution time of both the best and worst scaling factor and crossover rate combinations of each of the five problems. The results are in milliseconds (ms) and in generations per millisecond (G/ms).

	F_1	F_2	F_4	F_6	F_9
Best (ms):	38	33	35	54	85
Best (G/ms):	2.03	3.82	3.74	5.65	3.81
Worst (ms):	1234	1542	1506	1252	2175
Worst (G/ms):	8.10	6.49	6.64	7.99	4.60

Table 4.6: Running time details for the five problems.

4.3.7 Unimodal

When attempting to solve unimodal problems (F_1, F_2, F_4), the *DE/best-to-next/1* mutation scheme yielded an impressive mean quality measure difference of $+0.49$. Additionally, the mean values for the mean values for the two control parameters saw no change. The results can be seen in Table 4.7.

Mean F	:	0.50
Mean Cr	:	0.70
Mean Δ_{Q_m}	:	$+0.49$
Mean Δ_F	:	$+0.00$
Mean Δ_{Cr}	:	$+0.00$

Table 4.7: Unimodal performance summary.

4.3.8 Multimodal

Solving multimodal problems (F_6, F_9) returned a mean quality measure difference of $+0.26$ and a mean scaling factor difference of $+0.20$. More specifically, the *DE/best-to-next/1* mutation scheme returned a Δ_{Q_m} of $+0.44$ for the very difficult F_6 problem, but only managed a $+0.09$ improvement for the F_9 problem. The results are listed in Table 4.8.

Mean F	: 0.60
Mean Cr	: 0.50
Mean Δ_{Q_m}	: $+0.26$
Mean Δ_F	: $+0.20$
Mean Δ_{Cr}	: $+0.00$

Table 4.8: Multimodal performance summary.

4.3.9 Seperable

When attempting to solve seperable problems (F_1, F_9), an overall improvement of $+0.29$ in quality measure difference was recorded. Additionally, the mean scaling factor saw an increase of $+0.20$. The results are shown in Table 4.9.

Mean F	: 0.40
Mean Cr	: 0.40
Mean Δ_{Q_m}	: $+0.29$
Mean Δ_F	: $+0.20$
Mean Δ_{Cr}	: $+0.00$

Table 4.9: Seperable performance summary.

4.3.10 Non-seperable

The *DE/best-to-next/1* returned a mean quality measure improvement of $+0.47$ when solving non-separable problems (F_2, F_4, F_6). Also, there was no difference in the mean scaling factor and crossover rate values. The results are listed in Table 4.10.

Mean F	: 0.50
Mean Cr	: 0.80
Mean Δ_{Q_m}	: $+0.47$
Mean Δ_F	: $+0.00$
Mean Δ_{Cr}	: $+0.00$

Table 4.10: Non-seperable performance summary.

4.4 Conclusion

When comparing the classic mutation scheme, *DE/rand/1*, and the *DE/best-to-next/1*, an overall mean quality measure improvement of $+0.40$ was recorded. In fact, the first four problems had an improvement of more than $+0.40$, but the F_9 problem yielded an improvement of only $+0.09$. These results are shown in Table 4.11.

Mean F	:	0.50
Mean Cr	:	0.60
Mean Δ_{Q_m}	:	$+0.40$
Mean Δ_F	:	$+0.10$
Mean Δ_{Cr}	:	-0.10

Table 4.11: Overall performance summary.

Another noteworthy aspect was the slight shift in the mean scaling factor and crossover rate values. It seems that the *DE/best-to-next/1/bin* variant is more efficient with higher scaling factor and lower crossover rate values than the ones recorded with *DE/rand/1/bin*.

Bibliography

[1] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," 2005.

[2] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

Chapter 5

Conclusions

In the scientific community, the field of mathematical optimization remains an "open problem," with scientists from various disciplines attempting to find methods which will address this issue. In an attempt to deal with this problem, the Differential Evolution (DE) algorithm was developed.

DE emerged more than a decade ago as an extremely simple algorithm for dealing with the complex problem of mathematical optimization[1;2], and until today remains an active research topic within the Evolutionary Computation (EC) community. Since its inception, DE has been applied to various real-world problems, such as image processing[3;4], scheduling[5], optimal design[6] and community detection[7], and performs exceptionally against discrete, discontinuous, temporally changeable or highly noisy problems. DE is an agent-based stochastic solution optimizer for real-valued problems. Inspired by nature, DE simulates Darwinian evolution and genetics, attempting to exploit nature's tried-and-true optimization abilities.

This report attempted to provide the reader with a thorough analysis of the DE algorithm, while also testing its efficacy and addressing its shortcomings. More specifically, Chapter 1 provides an overview of the mathematical optimization field and the DE algorithm; Chapter 2 provides a detailed analysis of the stages and concepts of DE; Chapter 3 tested DE against problems from the CEC 2005 Test Suite[8] and a conclusion is reached in regards to the proper tuning of the DE control parameters; Chapter 4 introduced the *DE/best-to-next/1* mutation scheme in an attempt to provide a deterministic alternative to the plethora of stochastic mutation schemes.

Chapter 3 has shown that even though there exists no ideal combination of the scaling factor, F, crossover rate, Cr, and population size, NP, control parameters, assigning any of the scaling factor and crossover rate combinations listed in Table 3.13 should at the very least yield good performance for solving any kind of objective function. Furthermore, Chapter 4 has demonstrated that the use of the deterministic *DE/best-to-next/1* mutation scheme produced an overall improvement of $+0.40$ in quality measure, Q_m, compared to the the state-of-the-art *DE/rand/1* mutation scheme, as seen in Table 4.11.

Bibliography

[1] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *ICSI, USA*, 1995.

[2] R. Storn and K. Price, "Minimizing the real functions of the icec 1996 contest by differential evolution," *Proc. IEEE Int. Conf. Evol. Comput.*, pp. 842–844, 1996.

[3] I. D. Falco, A. D. Cioppa, and A. Tarantino, "Automatic classification of handsegmented image parts with differential evolution," *EvoWorkshops 2006*, pp. 403–414, 2006.

[4] M. Omran, A. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," *Proceedings of IEEE Congress on Evolutionary Computation*, vol. 2, pp. 966–973, 2005.

[5] A. Nearchou and S. Omirou, "Differential evolution for sequencing and scheduling optimization," *Journal of Heuristics*, no. 12, pp. 395–411, 2006.

[6] B. Babu and S. Munawar, "Differential evolution strategies for optimal design of shell-and-tube heat exchangers," *Chemical Engineering Science*, vol. 62, no. 14, 2007.

[7] G. Jia, Z. Cai, M. Musolesi, Y. Wang, D. Tennant, R. Weber, J. Heath, and S. He, "Community detection in social and biological networks using differential evolution," *Learning and Intelligent OptimizatioN Conference*, 2012.

[8] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," 2005.

Bibliography

[1] R. Storn and K. Price, "Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces," *ICSI, USA*, 1995.

[2] R. Storn and K. Price, "Minimizing the real functions of the icec 1996 contest by differential evolution," *Proc. IEEE Int. Conf. Evol. Comput.*, pp. 842–844, 1996.

[3] I. D. Falco, A. D. Cioppa, and A. Tarantino, "Automatic classification of handsegmented image parts with differential evolution," *EvoWorkshops 2006*, pp. 403–414, 2006.

[4] M. Omran, A. Engelbrecht, and A. Salman, "Differential evolution methods for unsupervised image classification," *Proceedings of IEEE Congress on Evolutionary Computation*, vol. 2, pp. 966–973, 2005.

[5] A. Nearchou and S. Omirou, "Differential evolution for sequencing and scheduling optimization," *Journal of Heuristics*, no. 12, pp. 395–411, 2006.

[6] B. Babu and S. Munawar, "Differential evolution strategies for optimal design of shell-and-tube heat exchangers," *Chemical Engineering Science*, vol. 62, no. 14, 2007.

[7] G. Jia, Z. Cai, M. Musolesi, Y. Wang, D. Tennant, R. Weber, J. Heath, and S. He, "Community detection in social and biological networks using differential evolution," *Learning and Intelligent OptimizatioN Conference*, 2012.

[8] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[9] P. Suganthan, N. Hansen, J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization," 2005.

[10] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[11] R. Storn and K. Price, "Differential evolution: A simple evolutionary strategy for fast optimization," *Dr. Dobb's Journal*, vol. 22, no. 4, pp. 18–24, 1997.

[12] R. Gämperle, S. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," *Advanced in Intelligent Systems, Fuzzy Systems, Evolutionary Computation, WSEAS Press, Interlaken, Switzerland*, pp. 293–298, 2002.

[13] J. Rönkkönen, S. Kukkonen, and K. Price, "Real-parameter optimization with differential evolution," *IEEE Congress on Evolutionary Computation*, pp. 506–513, 2005.

[14] G. Jeyakumar and C. Shanmugavelayutham, "Convergence analysis of differential evolution variants on unconstrained global optimization functions," *International Journal of Artificial Intelligence and Applications (IJAIA)*, vol. 2, no. 2, 2011.

[15] R. Storn, "On the usage of differential evolution for function optimization," *Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), IEEE, Berkeley*, pp. 519–523, 1996.

[16] F. Ahlers, W. D. Carlo, C. Fleiner, L. Godwin, M. Keenan, R. Nath, A. Neumaier, J. Phillips, K. Price, R. Storn, P. Turney, F.-S. Wang, J. V. Zandt, H. Geldon, P. Gauden, C. Brauer, K. Shivaram, and D. Novikov, "Differential evolution website." http://www.icsi.berkeley.edu/~storn/code.html. [Online; last accessed on February 20 2012].

[17] D. Zaharie, "A comparative analysis of crossover variants in differential evolution," *Proceedings of the International Multiconference on Computer Science and Information Technology*, pp. 171–181, 2007.

[18] V. Feoktistov and S. Janaqi, "Generalization of the strategies in differential evolution," *Proc. 18th IPDPS*, p. 165a, 2004.

[19] A. Qin, V. Huang, and P. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *IEEE Trans. Evol. Comput.*, vol. 13, no. 2, pp. 398–417, 2009.

[20] S. Das, A. Abraham, U. Chakraborty, and A. Konar, "Differential evolution using a neighborhood based mutation operator," *IEEE Trans. Evol. Comput.*, vol. 13, no. 3, pp. 526–553, 2009.

[21] K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2005.

[22] A. Sutton, M. Lunacek, and L. Whitley, "Differential evolution and non-separability: Using selective pressure to focus search," *Proc. 9th Annu. Conf. GECCO*, pp. 1428–1435, 2007.

[23] R. Storn, *Advances in Differential Evolution*. U. K. Chakraborty, Ed. Berlin, Germany: Springer, 2008.

Appendices

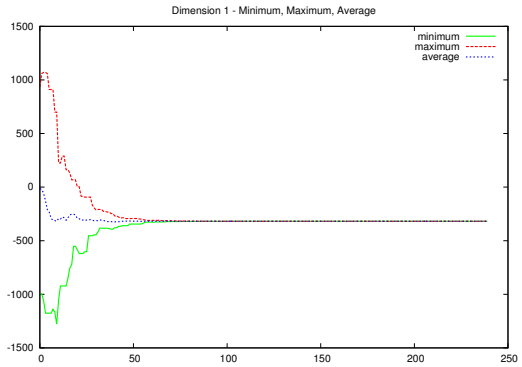
Appendix A

Extra Graphs

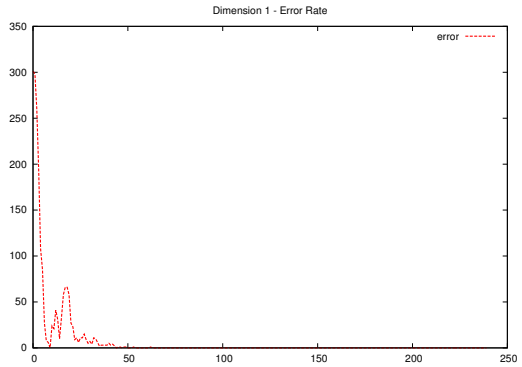
In the next pages, extra graphs for the five functions that were tested by the DET are found. These graphs give a more in-depth view on the convergence rate of DE or each of the two variants of the algorithm ran (*DE/rand/1/bin* and *DE/best-to-next/1/bin*), but they were too bulky to be included in the main text.

There are twenty graphs per problem: ten graphs which show the minimum, maximum and average values the population had at each generation, G, per dimension, D; and ten which show the error (i.e., the distance to the solution) at each generation, G, per dimension D.

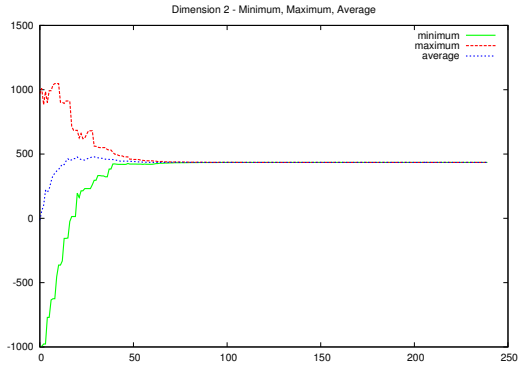
A.1 Problem F1 – DE/rand/1/bin



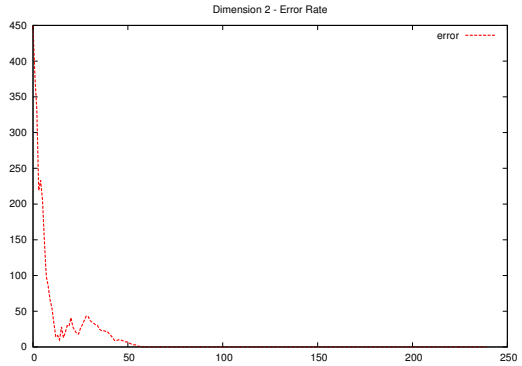
(a) $D1$ - Minimum, Maximum, Average



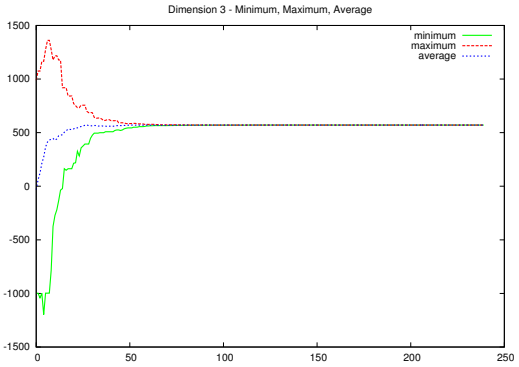
(b) $D1$ - Error



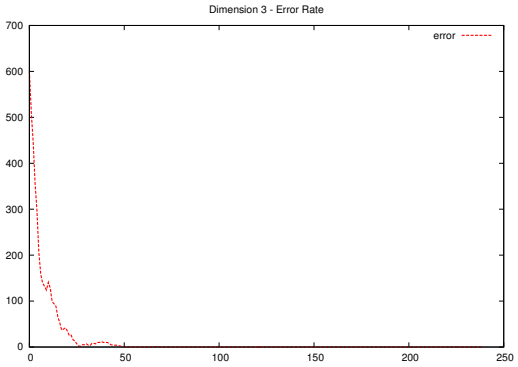
(c) $D2$ - Minimum, Maximum, Average



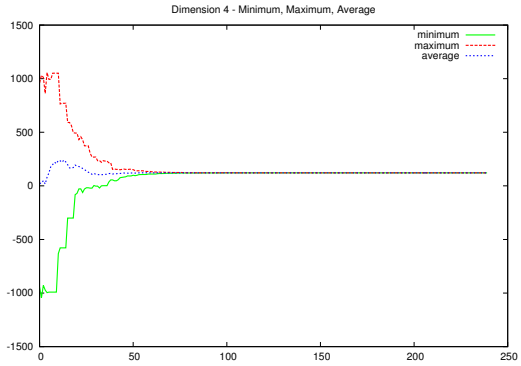
(d) $D2$ - Error



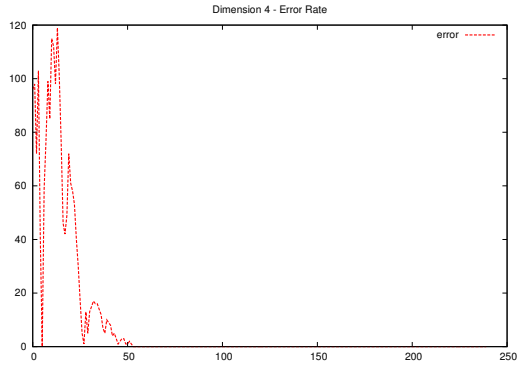
(e) $D3$ - Minimum, Maximum, Average



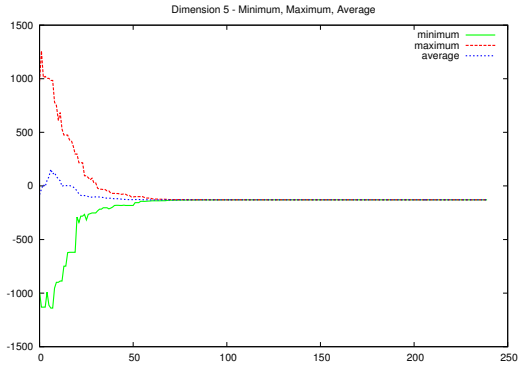
(f) $D3$ - Error



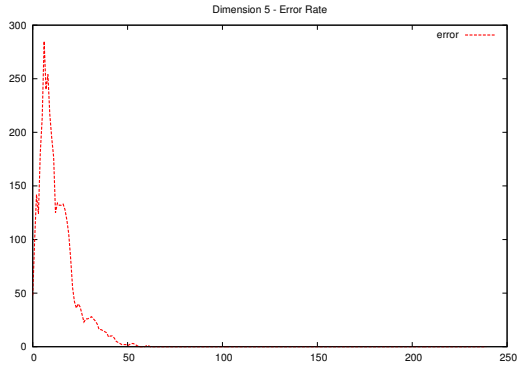
(g) $D4$ - Minimum, Maximum, Average



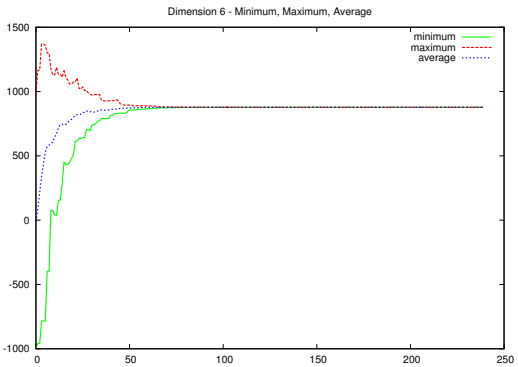
(h) $D4$ - Error



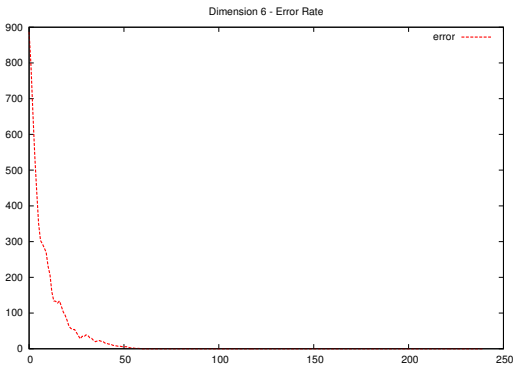
(i) $D5$ - Minimum, Maximum, Average



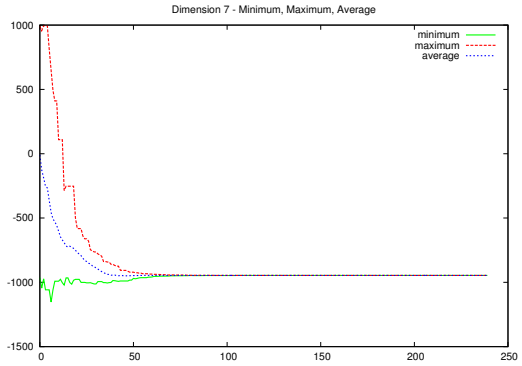
(j) $D5$ - Error



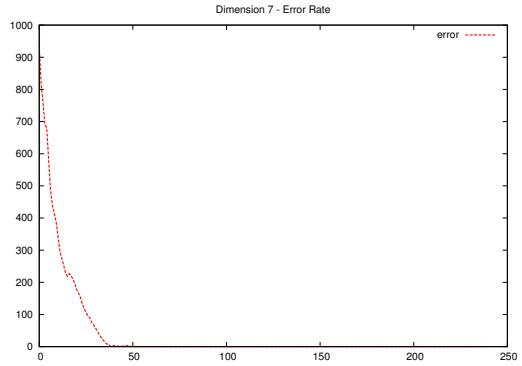
(k) $D6$ - Minimum, Maximum, Average



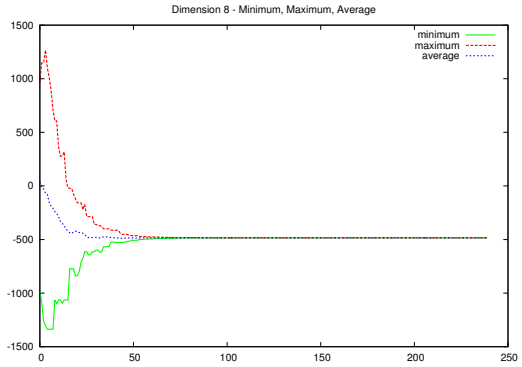
(l) $D6$ - Error



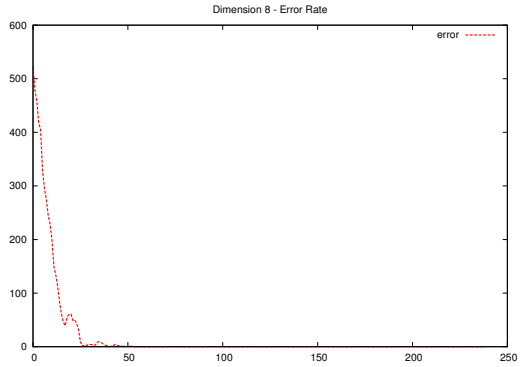
(m) *D7* - Minimum, Maximum, Average



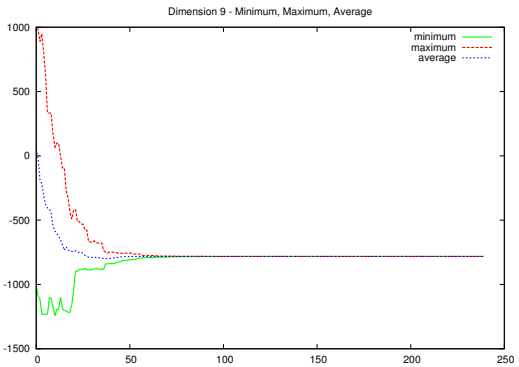
(n) *D7* - Error



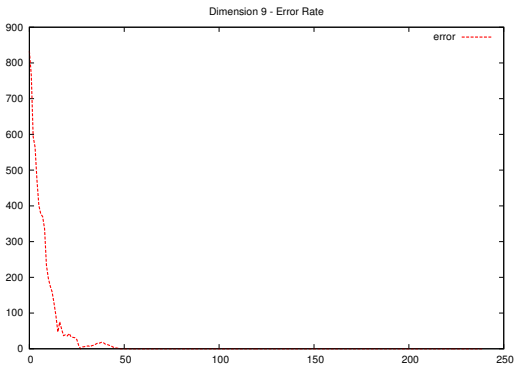
(o) *D8* - Minimum, Maximum, Average



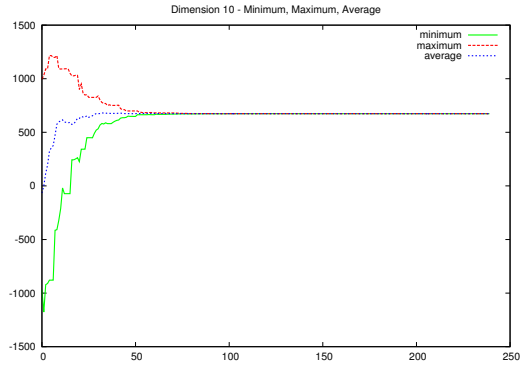
(p) *D8* - Error



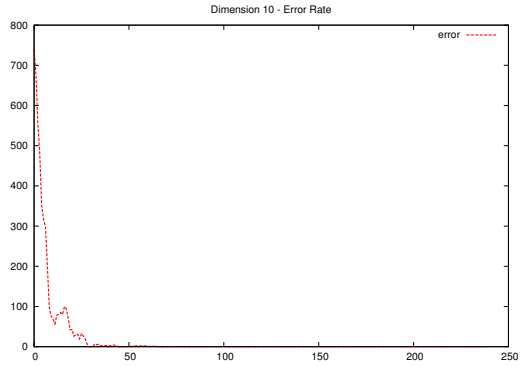
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



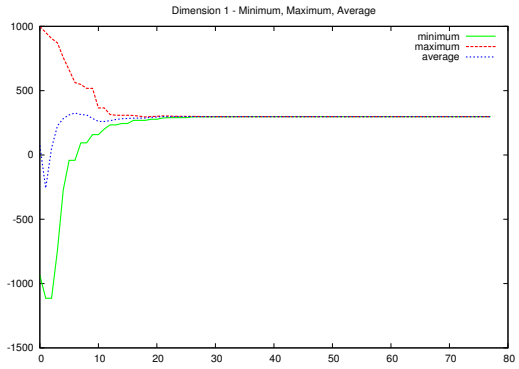
(s) $D10$ - Minimum, Maximum, Average



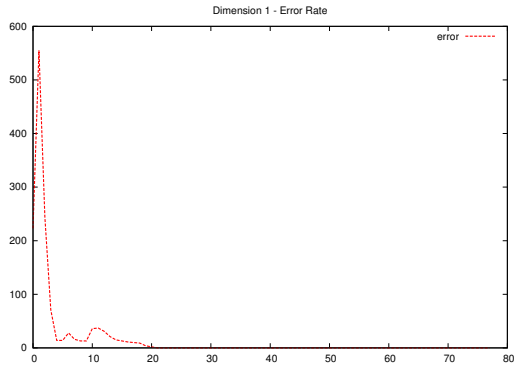
(t) $D10$ - Error

Figure A.1: Minimum, maximum, average, and error graphs produced by the DET for a typical run of a 10-D Shifted Sphere Function with control parameters $NP = 101$, $F = 0.20$ and $Cr = 0.70$.

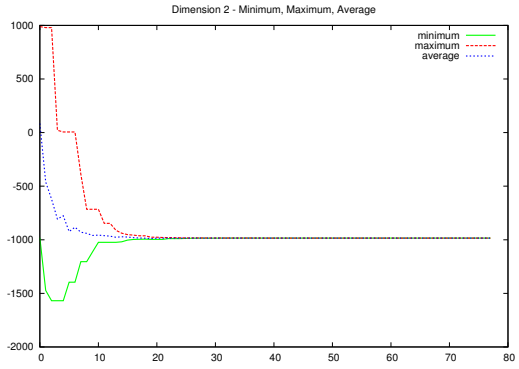
A.2 Problem F1 – DE/best-to-next/1/bin



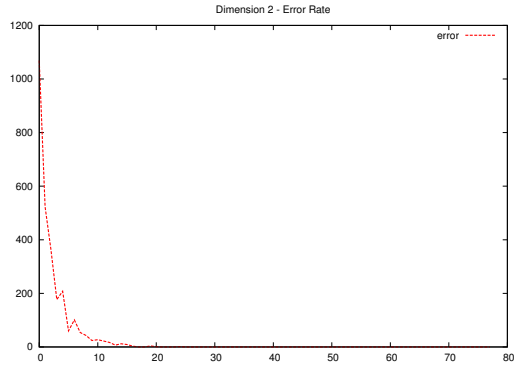
(a) $D1$ - Minimum, Maximum, Average



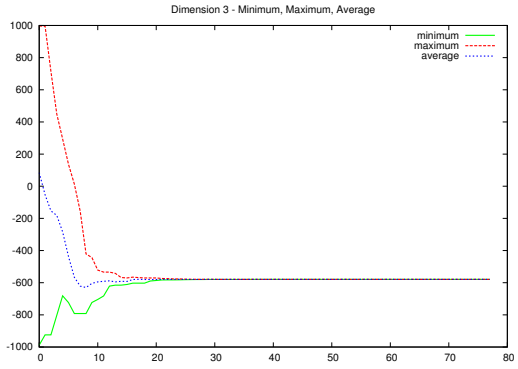
(b) $D1$ - Error



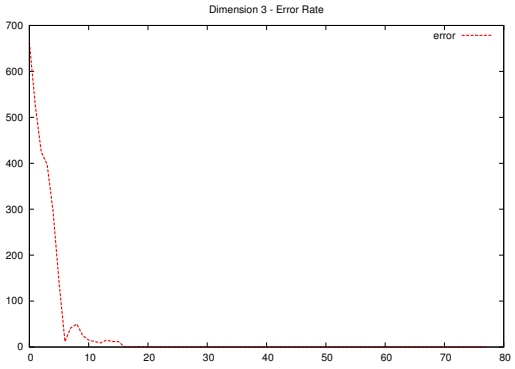
(c) $D2$ - Minimum, Maximum, Average



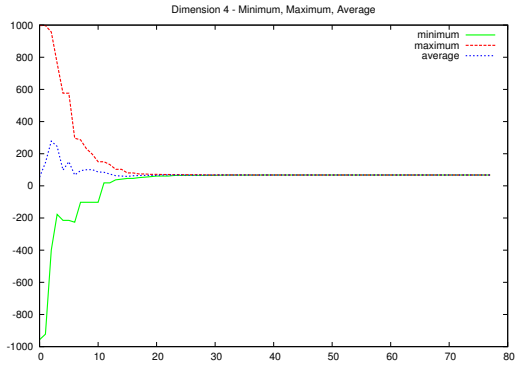
(d) $D2$ - Error



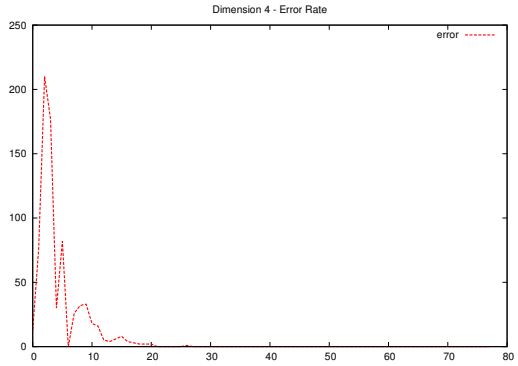
(e) $D3$ - Minimum, Maximum, Average



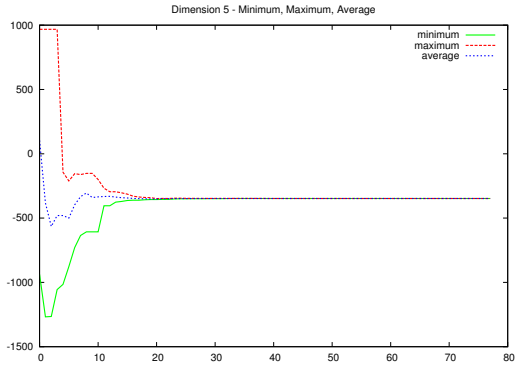
(f) $D3$ - Error



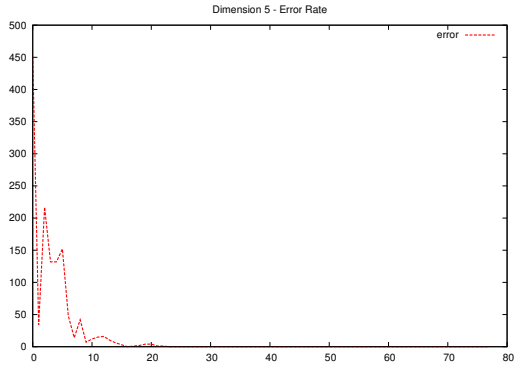
(g) $D4$ - Minimum, Maximum, Average



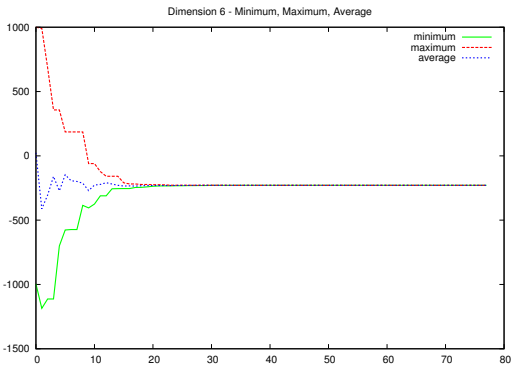
(h) $D4$ - Error



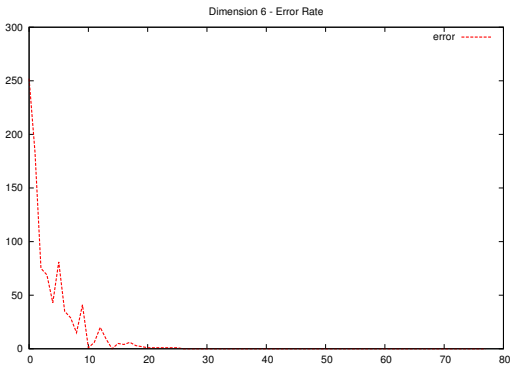
(i) $D5$ - Minimum, Maximum, Average



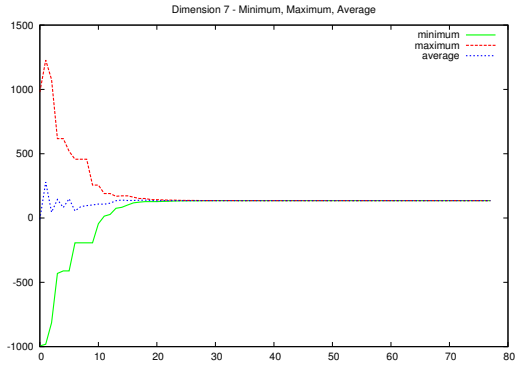
(j) $D5$ - Error



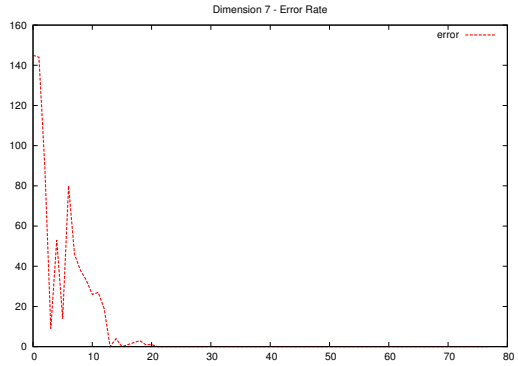
(k) $D6$ - Minimum, Maximum, Average



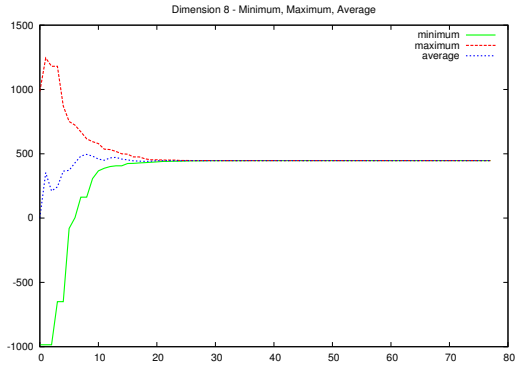
(l) $D6$ - Error



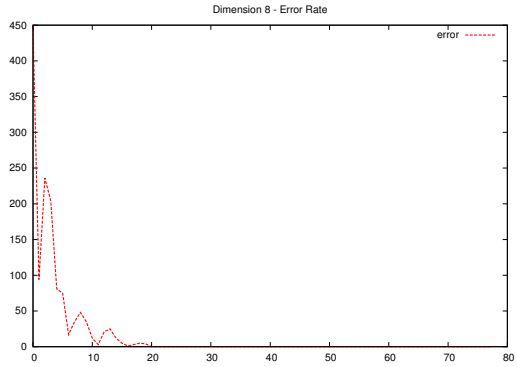
(m) *D7* - Minimum, Maximum, Average



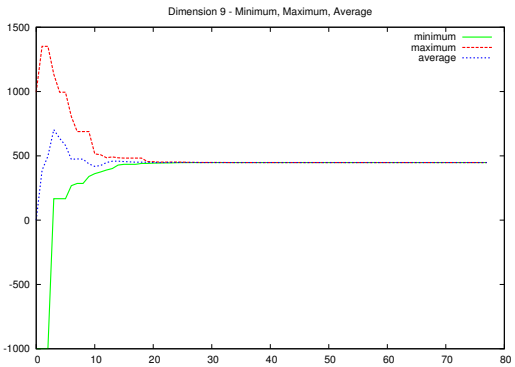
(n) *D7* - Error



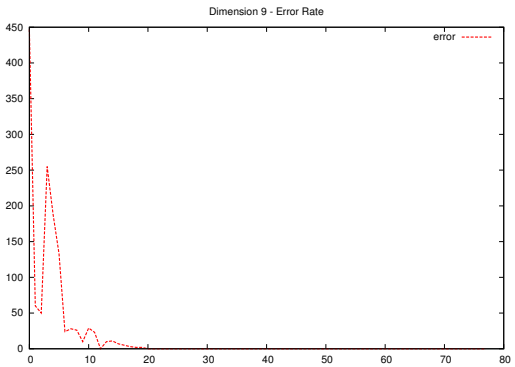
(o) *D8* - Minimum, Maximum, Average



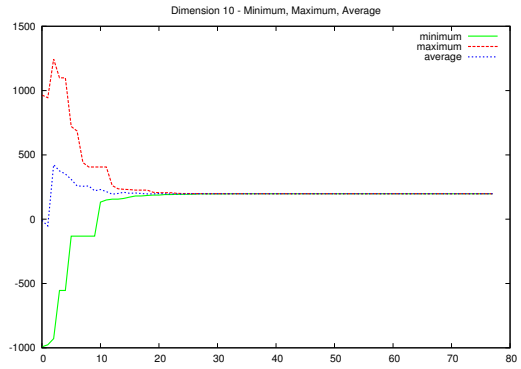
(p) *D8* - Error



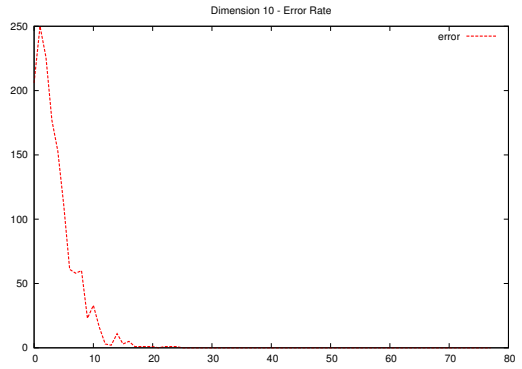
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



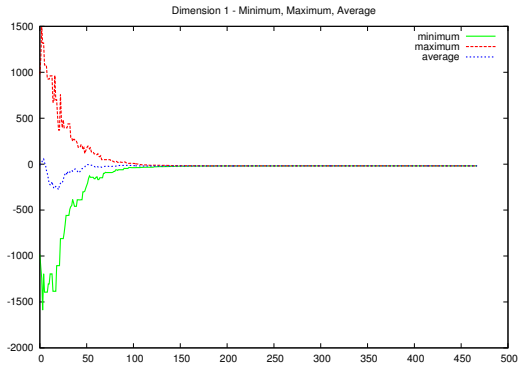
(s) $D10$ - Minimum, Maximum, Average



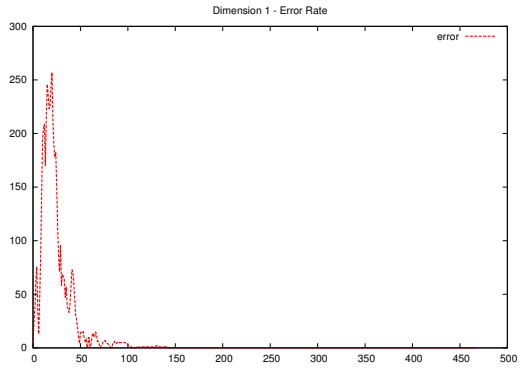
(t) $D10$ - Error

Figure A.2: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/best-to-next/1/bin* against a 10-D Shifted Sphere Function with control parameters $NP = 101$, $F = 0.40$ and $Cr = 0.70$.

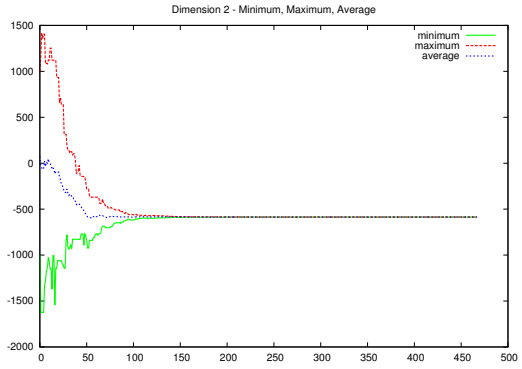
A.3 Problem F2 – DE/rand/1/bin



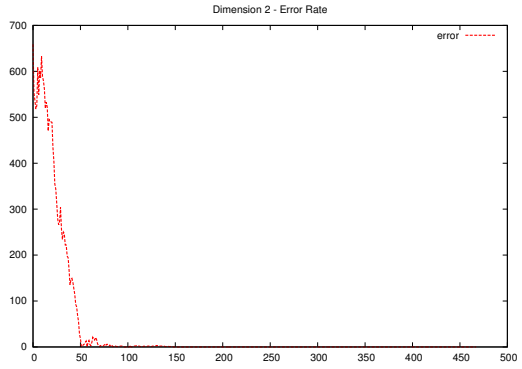
(a) $D1$ - Minimum, Maximum, Average



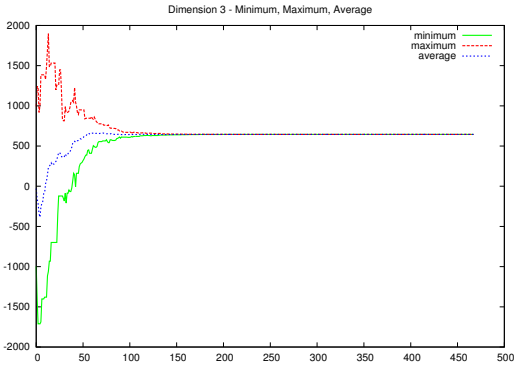
(b) $D1$ - Error



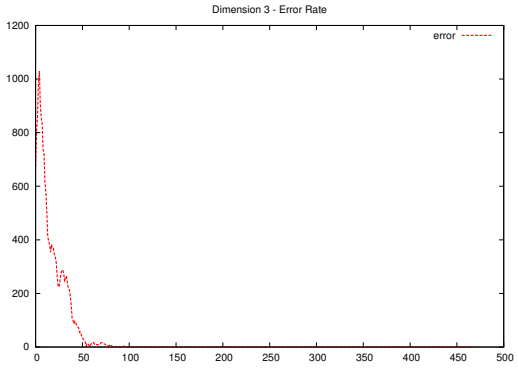
(c) $D2$ - Minimum, Maximum, Average



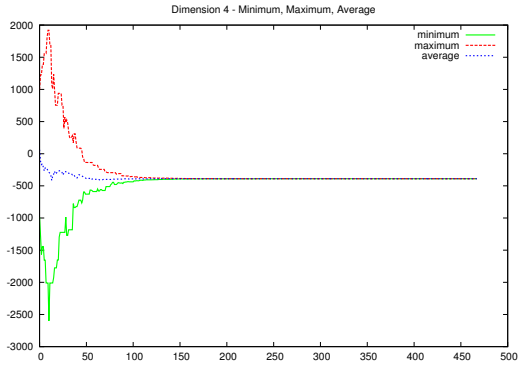
(d) $D2$ - Error



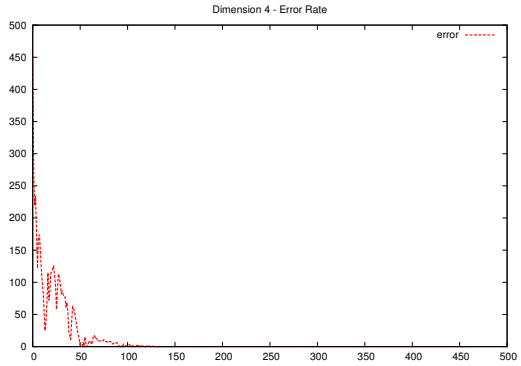
(e) $D3$ - Minimum, Maximum, Average



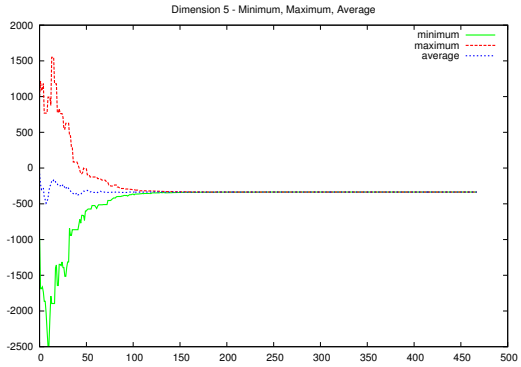
(f) $D3$ - Error



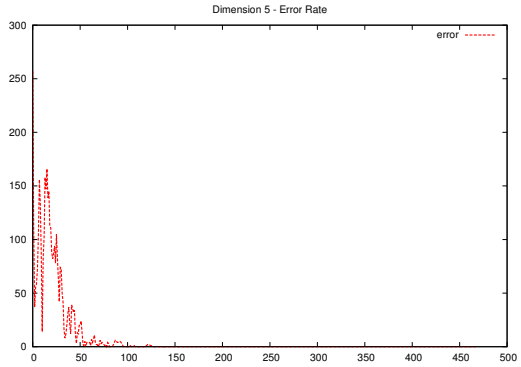
(g) $D4$ - Minimum, Maximum, Average



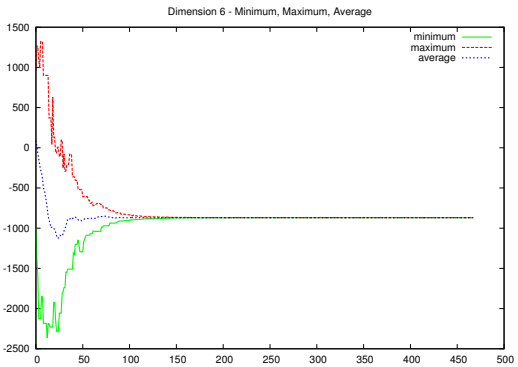
(h) $D4$ - Error



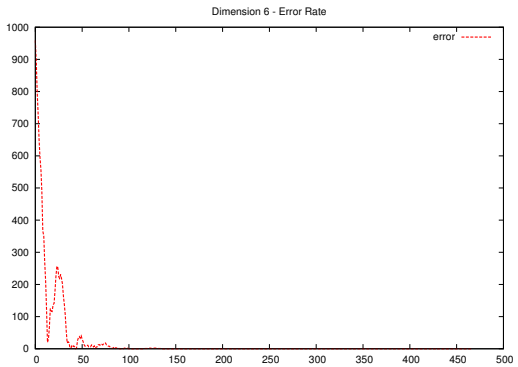
(i) $D5$ - Minimum, Maximum, Average



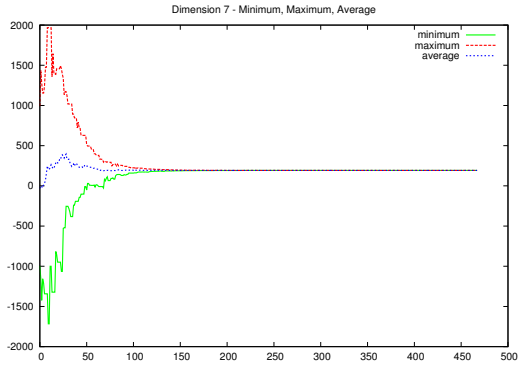
(j) $D5$ - Error



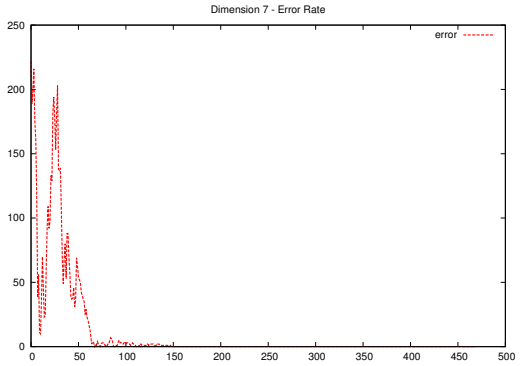
(k) $D6$ - Minimum, Maximum, Average



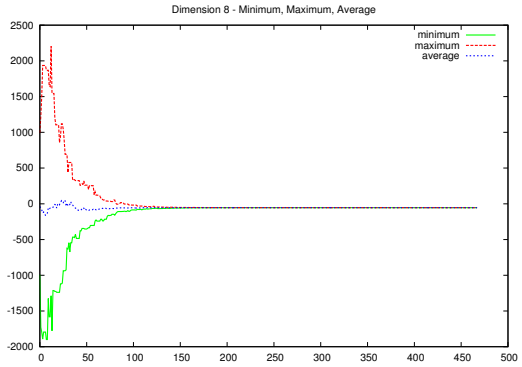
(l) $D6$ - Error



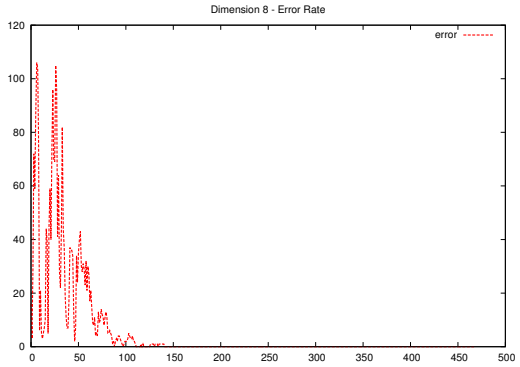
(m) *D7* - Minimum, Maximum, Average



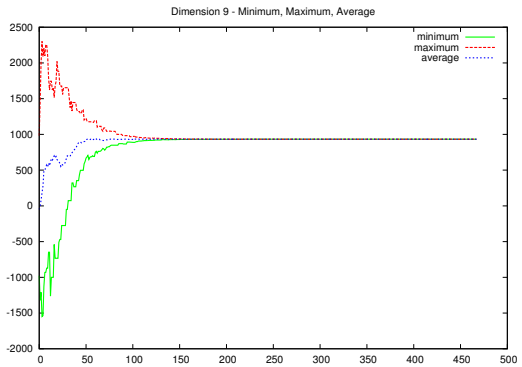
(n) *D7* - Error



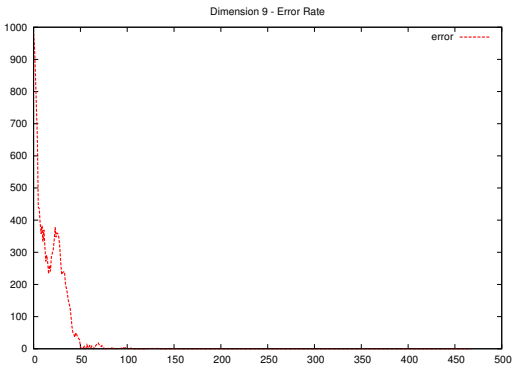
(o) *D8* - Minimum, Maximum, Average



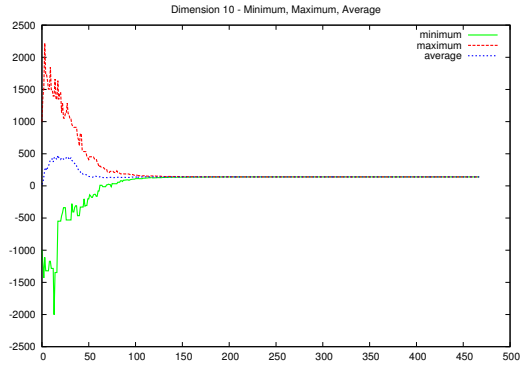
(p) *D8* - Error



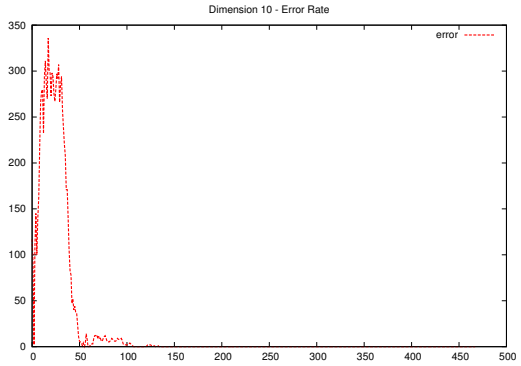
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



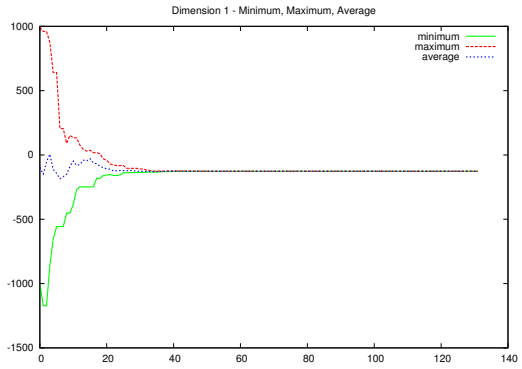
(s) $D10$ - Minimum, Maximum, Average



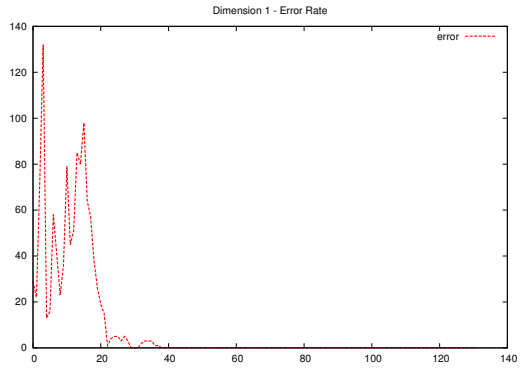
(t) $D10$ - Error

Figure A.3: Minimum, maximum, average, and error graphs produced by the DET for a typical run of a 10-D Shifted Schwefel's Problem with control parameters $NP = 101$, $F = 0.50$ and $Cr = 1.00$.

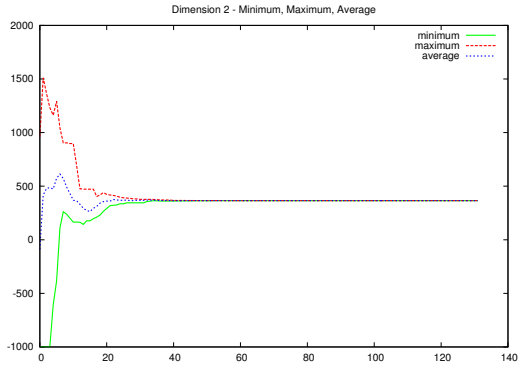
A.4 Problem F2 – DE/best-to-next/1/bin



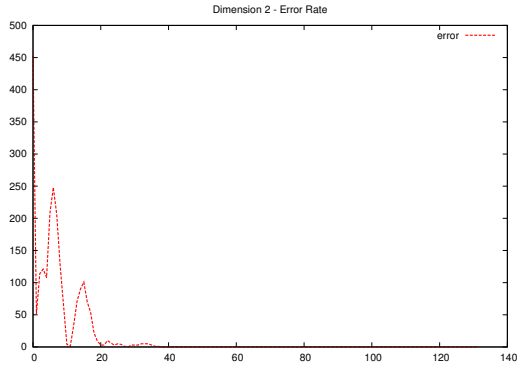
(a) $D1$ - Minimum, Maximum, Average



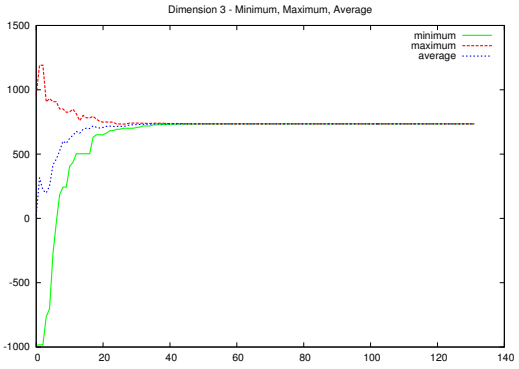
(b) $D1$ - Error



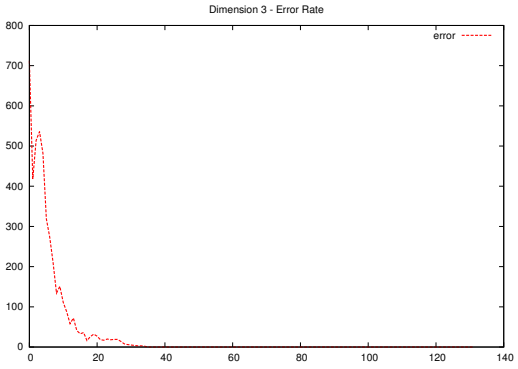
(c) $D2$ - Minimum, Maximum, Average



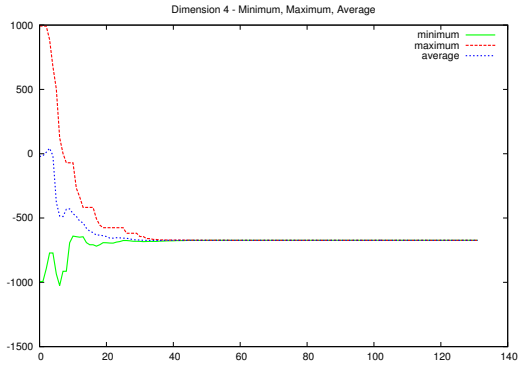
(d) $D2$ - Error



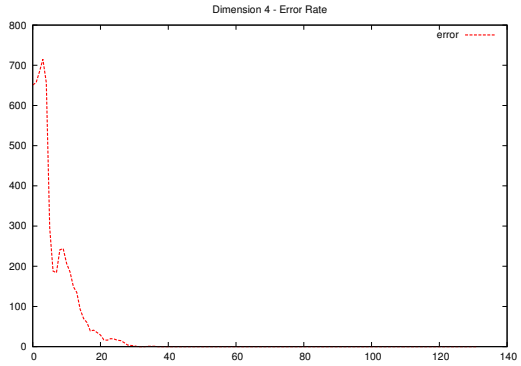
(e) $D3$ - Minimum, Maximum, Average



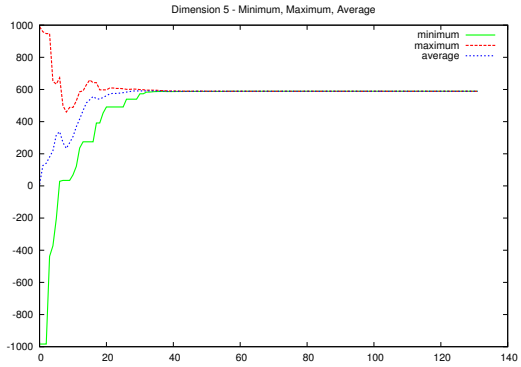
(f) $D3$ - Error



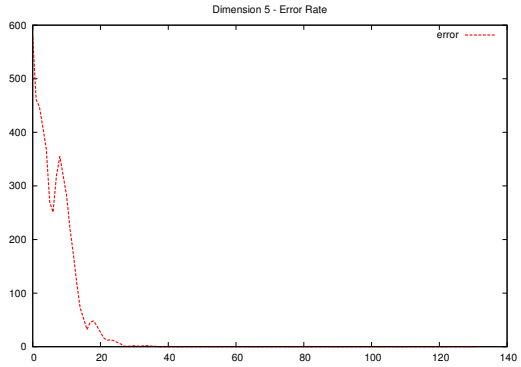
(g) $D4$ - Minimum, Maximum, Average



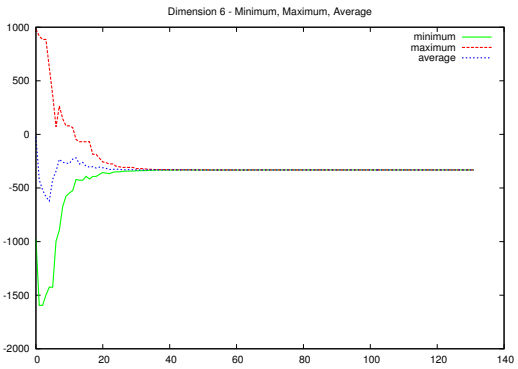
(h) $D4$ - Error



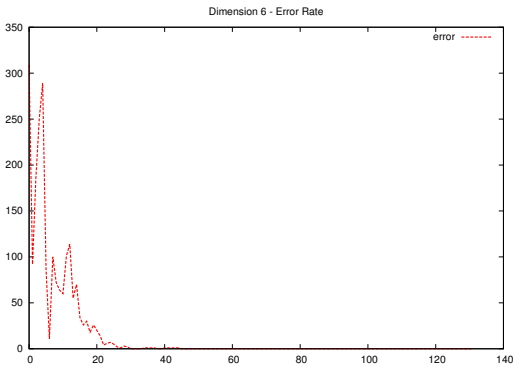
(i) $D5$ - Minimum, Maximum, Average



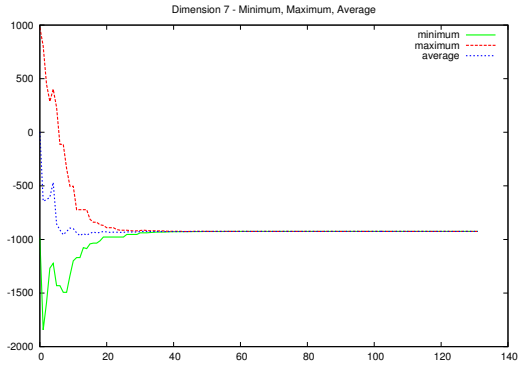
(j) $D5$ - Error



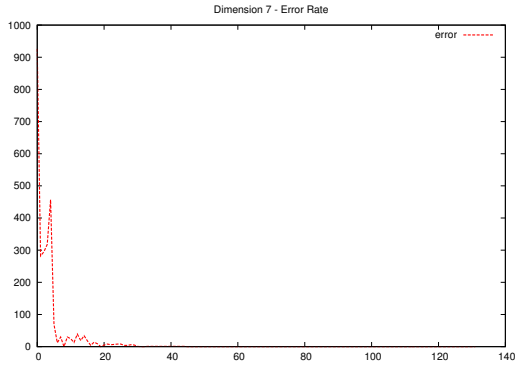
(k) $D6$ - Minimum, Maximum, Average



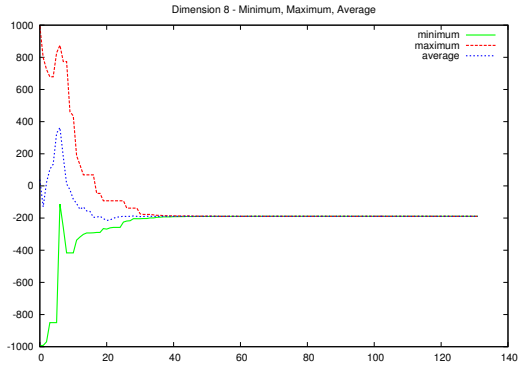
(l) $D6$ - Error



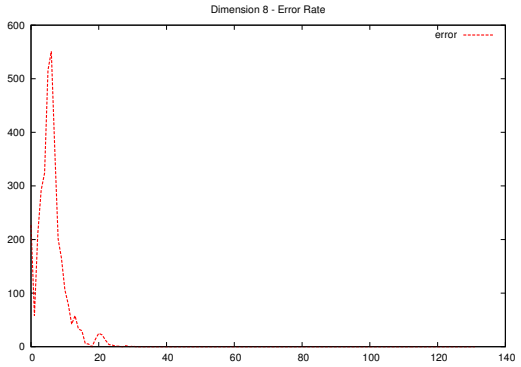
(m) *D7* - Minimum, Maximum, Average



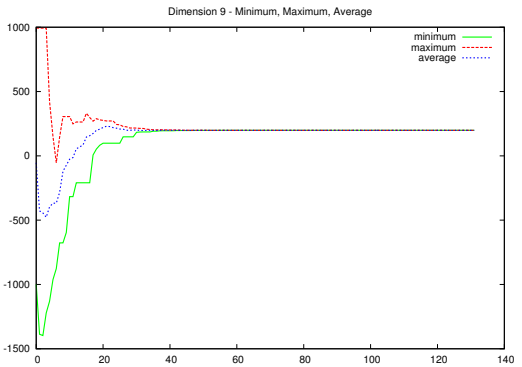
(n) *D7* - Error



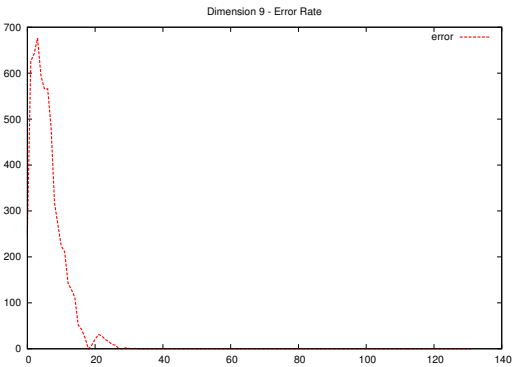
(o) *D8* - Minimum, Maximum, Average



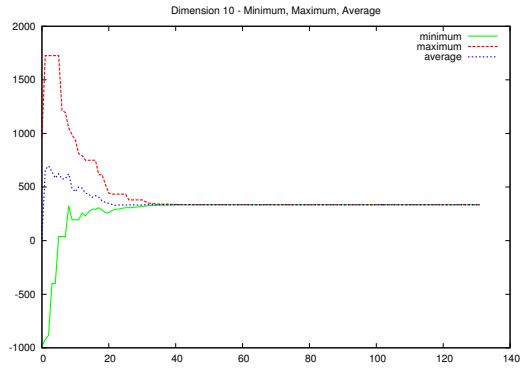
(p) *D8* - Error



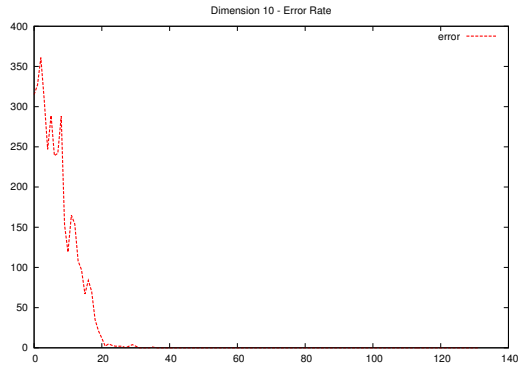
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



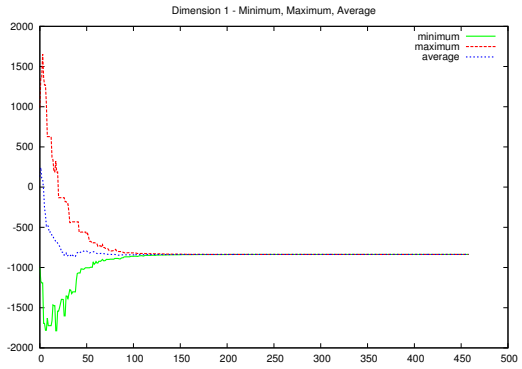
(s) $D10$ - Minimum, Maximum, Average



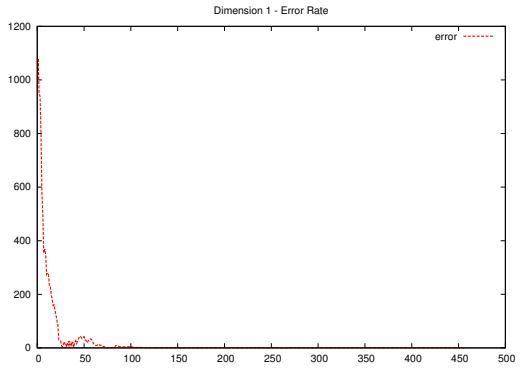
(t) $D10$ - Error

Figure A.4: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/best-to-next/1/bin* against a 10-D Shifted Schwefel's Problem with control parameters $NP = 101$, $F = 0.50$ and $Cr = 0.90$.

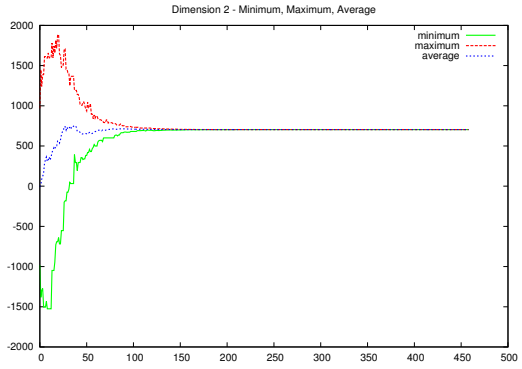
A.5 Problem F4 – DE/rand/1/bin



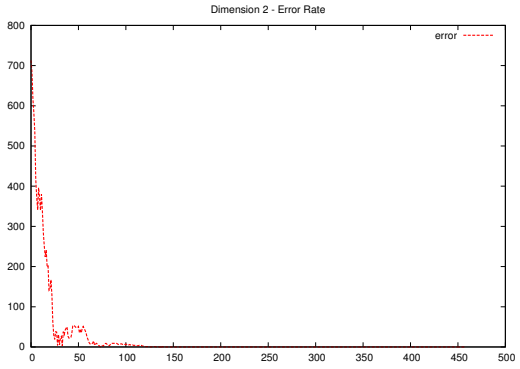
(a) $D1$ - Minimum, Maximum, Average



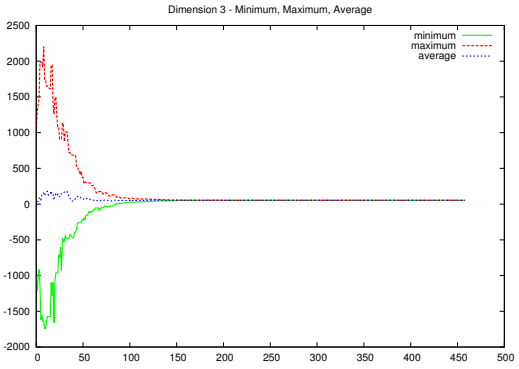
(b) $D1$ - Error



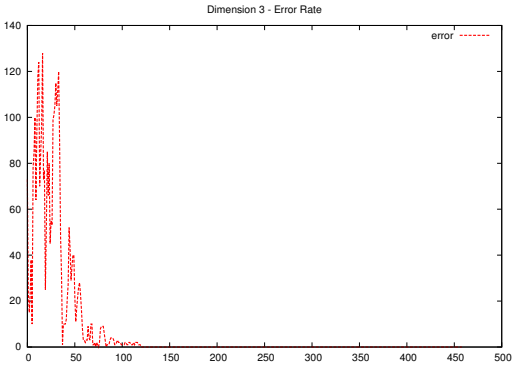
(c) $D2$ - Minimum, Maximum, Average



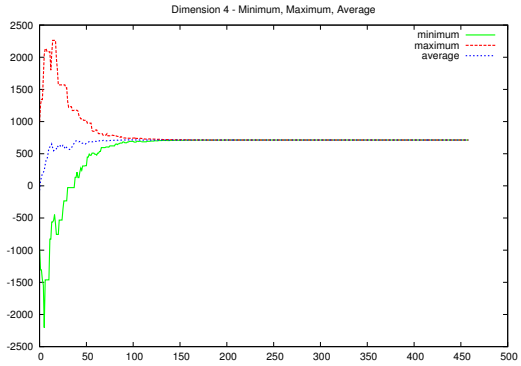
(d) $D2$ - Error



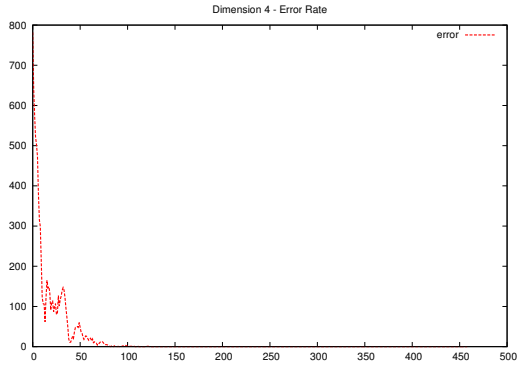
(e) $D3$ - Minimum, Maximum, Average



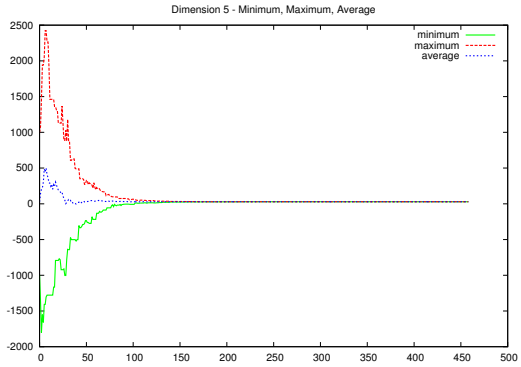
(f) $D3$ - Error



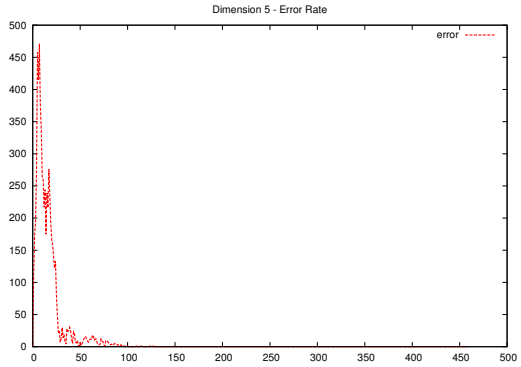
(g) $D4$ - Minimum, Maximum, Average



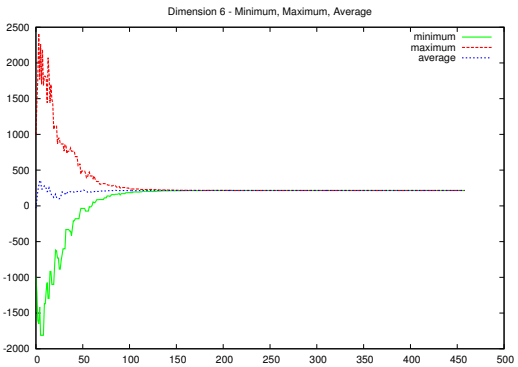
(h) $D4$ - Error



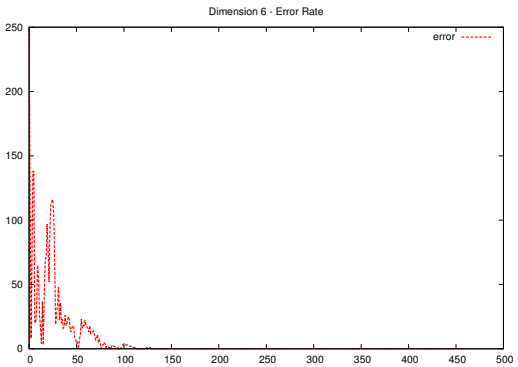
(i) $D5$ - Minimum, Maximum, Average



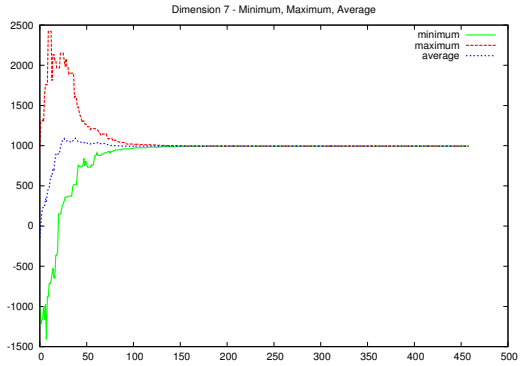
(j) $D5$ - Error



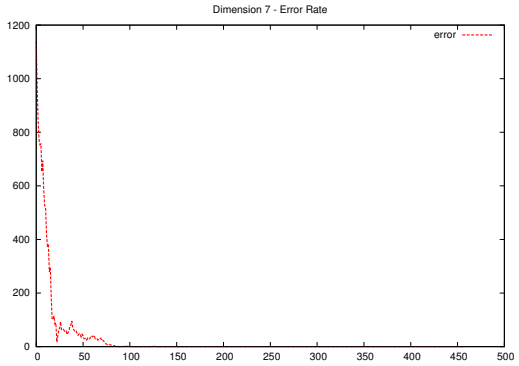
(k) $D6$ - Minimum, Maximum, Average



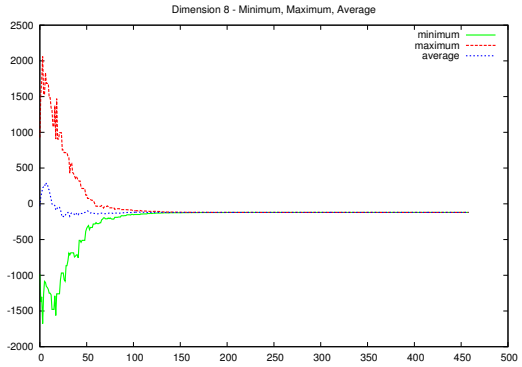
(l) $D6$ - Error



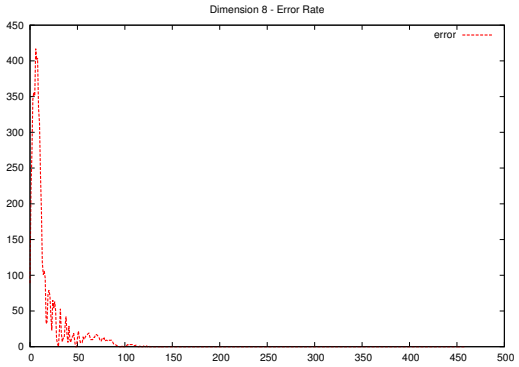
(m) *D7* - Minimum, Maximum, Average



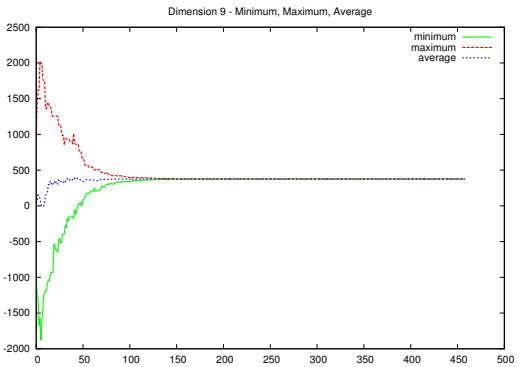
(n) *D7* - Error



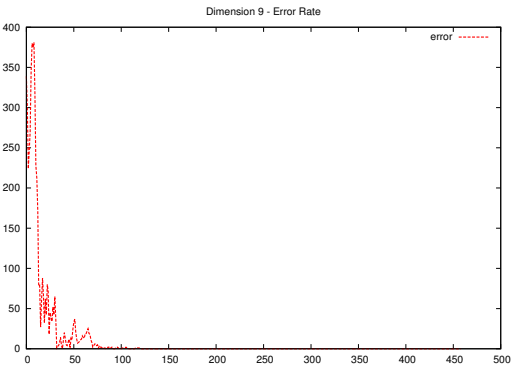
(o) *D8* - Minimum, Maximum, Average



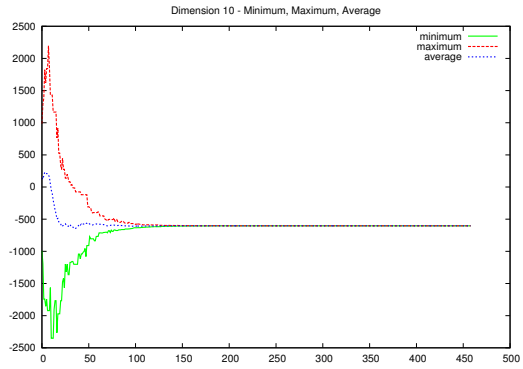
(p) *D8* - Error



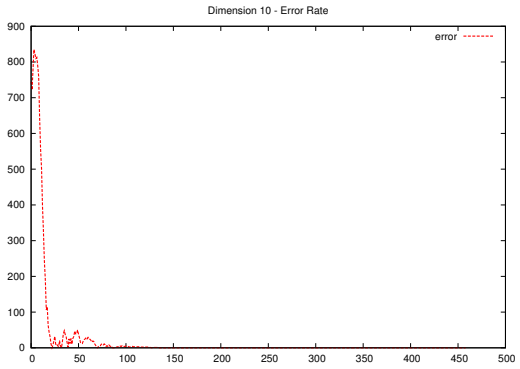
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



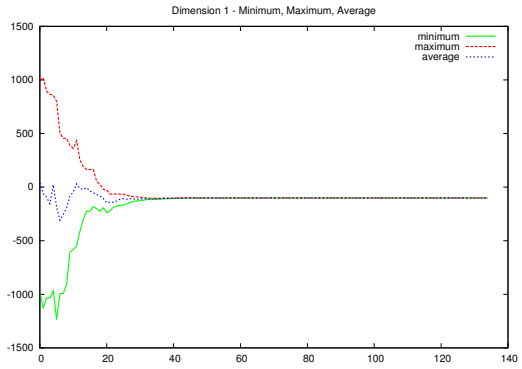
(s) $D10$ - Minimum, Maximum, Average



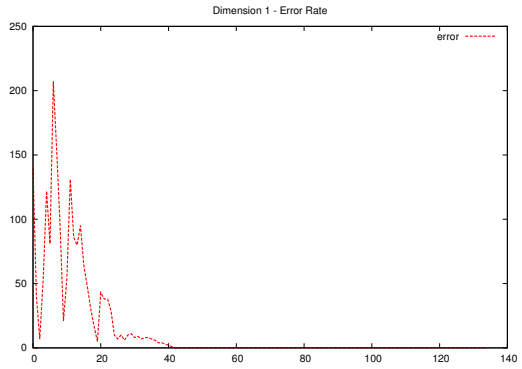
(t) $D10$ - Error

Figure A.5: Minimum, maximum, average, and error graphs produced by the DET for a typical run of a 10-D Shifted Schwefel's Problem with Noise in Fitness with control parameters $NP = 101$, $F = 0.50$ and $Cr = 1.00$.

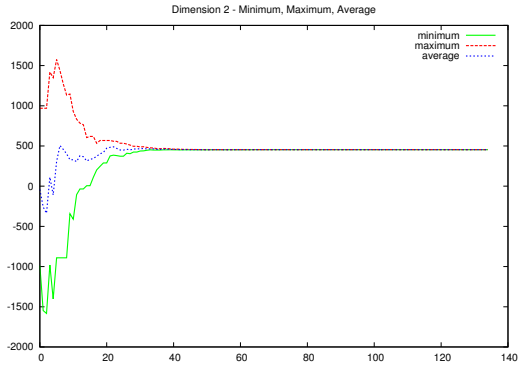
A.6 Problem F4 – DE/best-to-next/1/bin



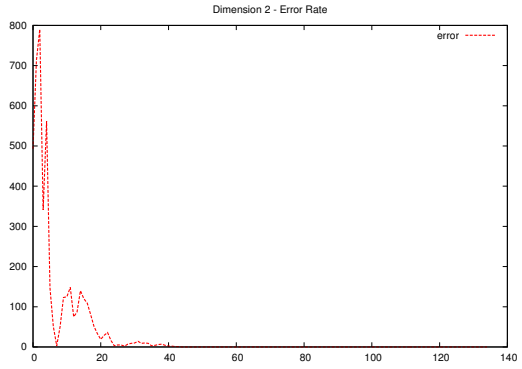
(a) $D1$ - Minimum, Maximum, Average



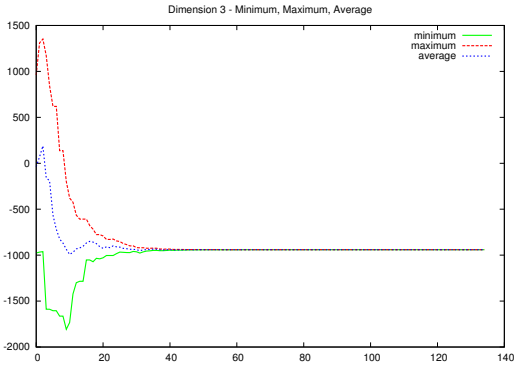
(b) $D1$ - Error



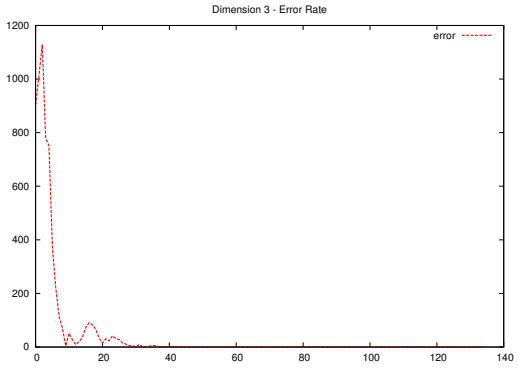
(c) $D2$ - Minimum, Maximum, Average



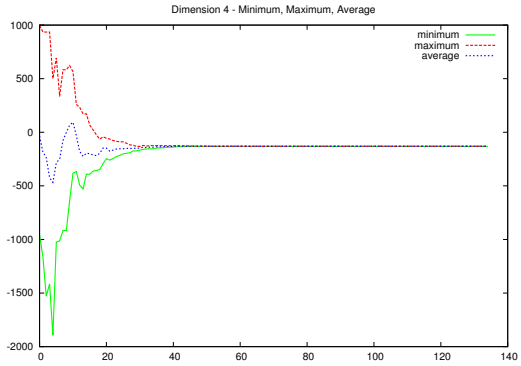
(d) $D2$ - Error



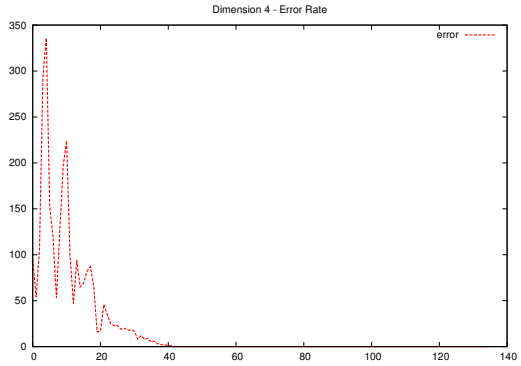
(e) $D3$ - Minimum, Maximum, Average



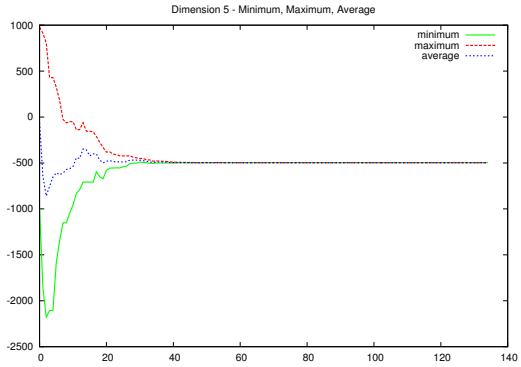
(f) $D3$ - Error



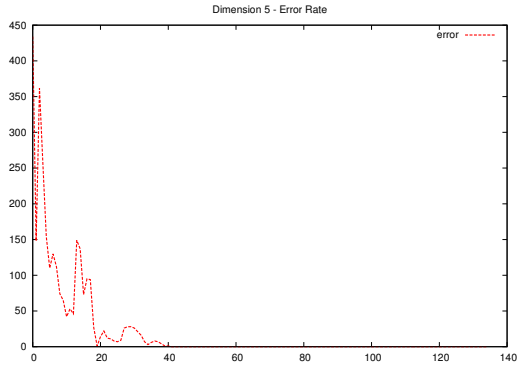
(g) $D4$ - Minimum, Maximum, Average



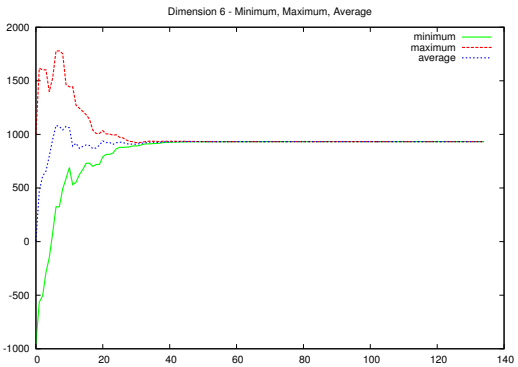
(h) $D4$ - Error



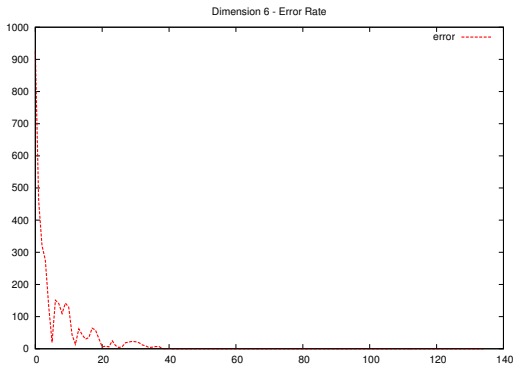
(i) $D5$ - Minimum, Maximum, Average



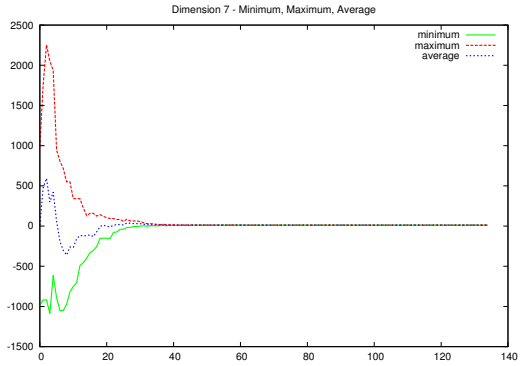
(j) $D5$ - Error



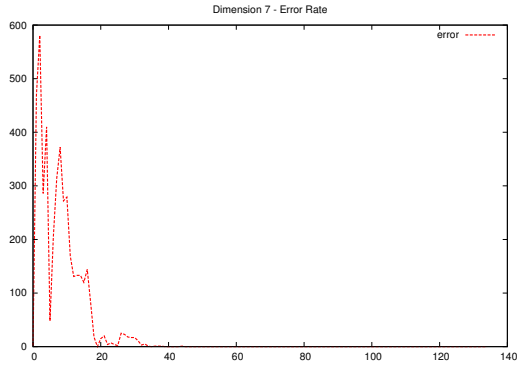
(k) $D6$ - Minimum, Maximum, Average



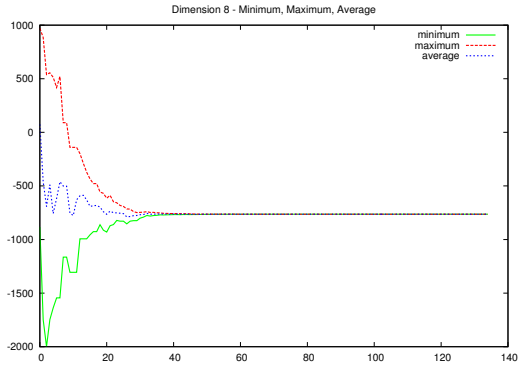
(l) $D6$ - Error



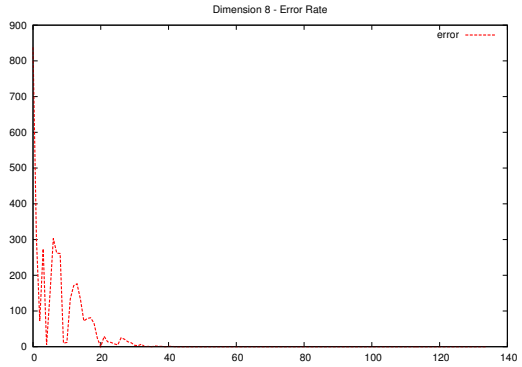
(m) *D7* - Minimum, Maximum, Average



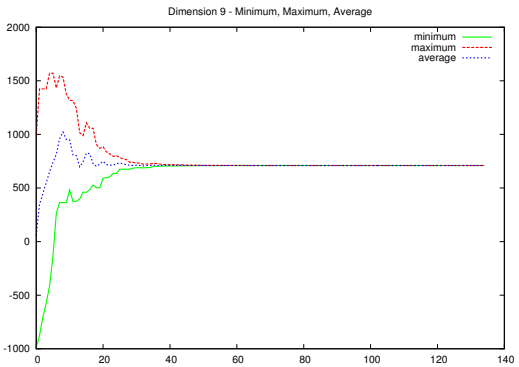
(n) *D7* - Error



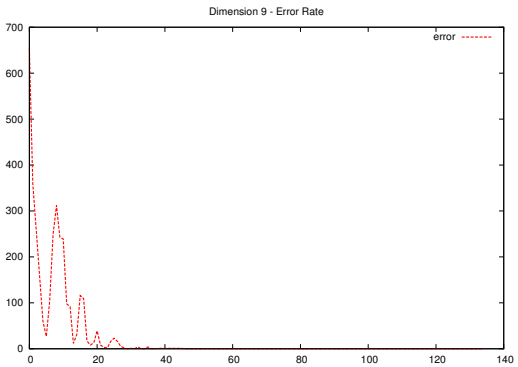
(o) *D8* - Minimum, Maximum, Average



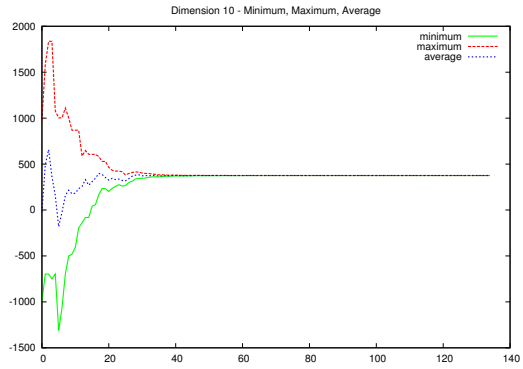
(p) *D8* - Error



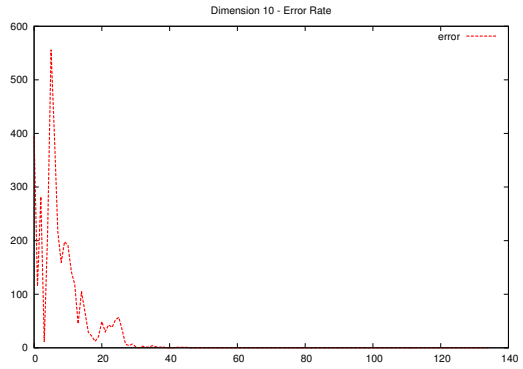
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



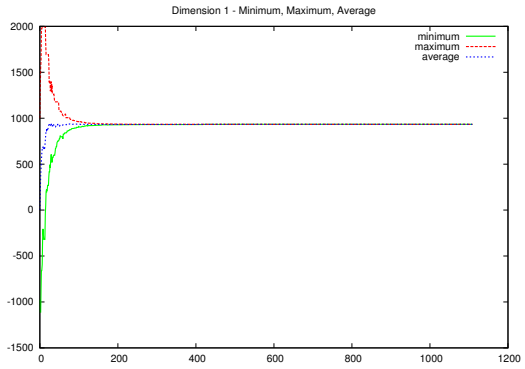
(s) $D10$ - Minimum, Maximum, Average



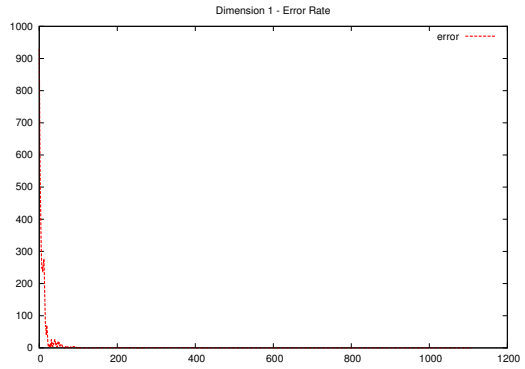
(t) $D10$ - Error

Figure A.6: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/best-to-next/1/bin* against a 10-D Shifted Schwefel's Problem with Noise in Fitness with control parameters $NP = 101$, $F = 0.60$ and $Cr = 1.00$.

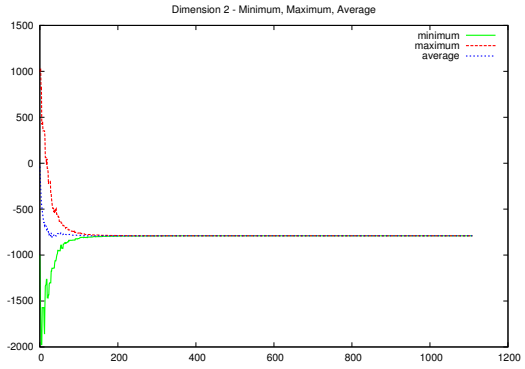
A.7 Problem F6 – DE/rand/1/bin



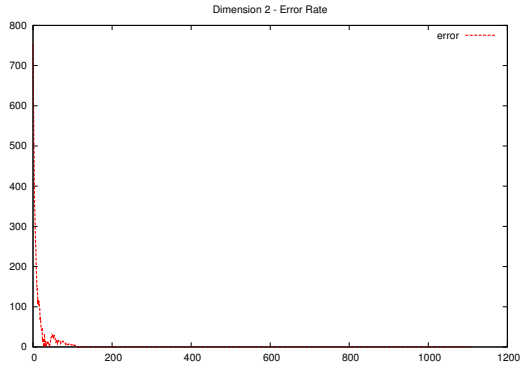
(a) $D1$ - Minimum, Maximum, Average



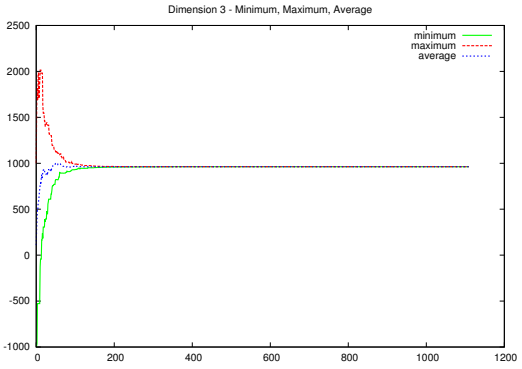
(b) $D1$ - Error



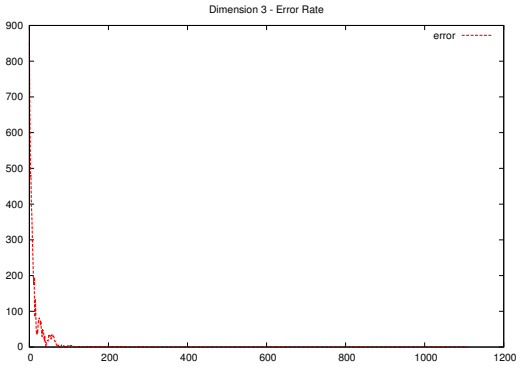
(c) $D2$ - Minimum, Maximum, Average



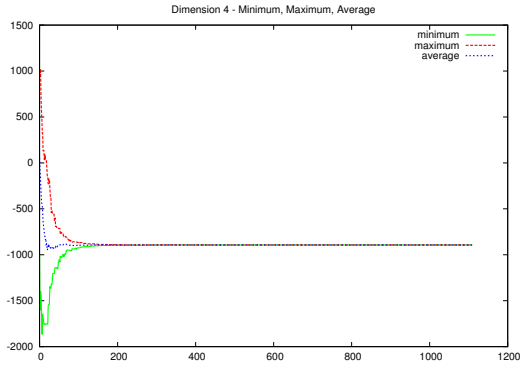
(d) $D2$ - Error



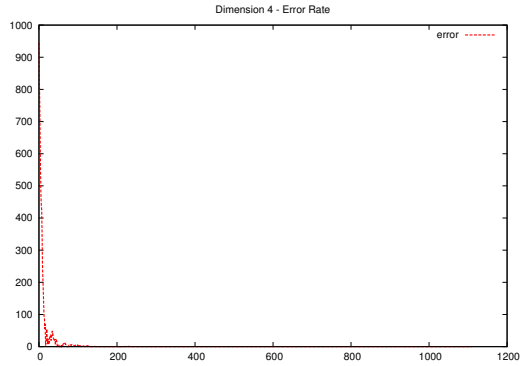
(e) $D3$ - Minimum, Maximum, Average



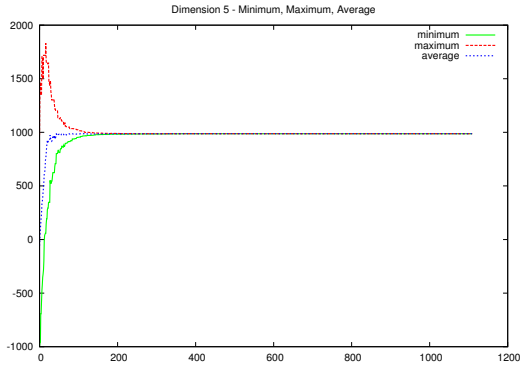
(f) $D3$ - Error



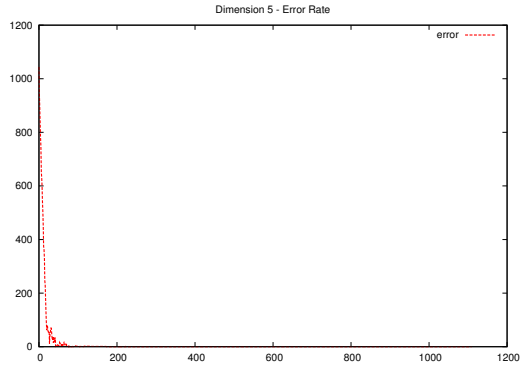
(g) $D4$ - Minimum, Maximum, Average



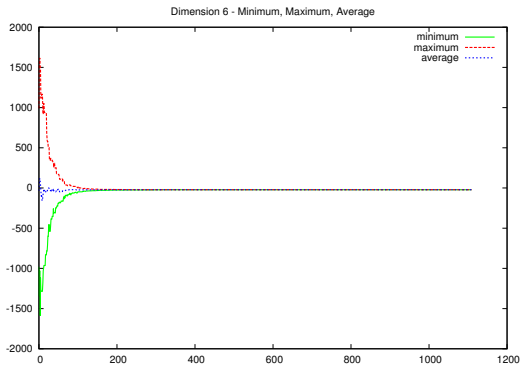
(h) $D4$ - Error



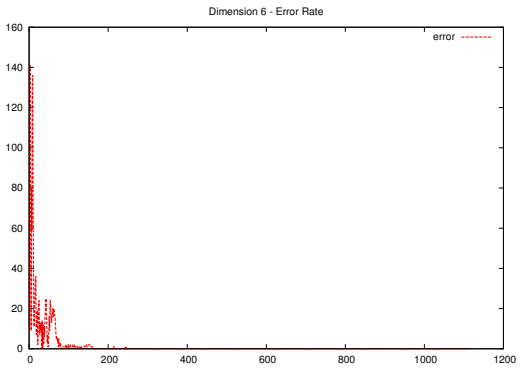
(i) $D5$ - Minimum, Maximum, Average



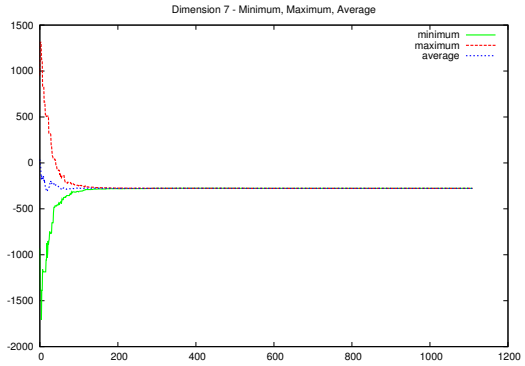
(j) $D5$ - Error



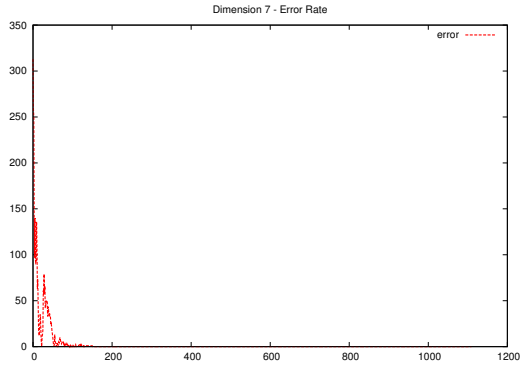
(k) $D6$ - Minimum, Maximum, Average



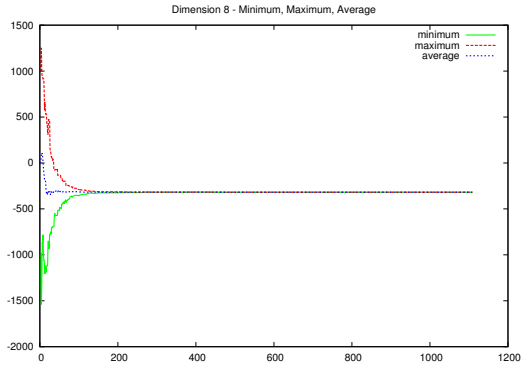
(l) $D6$ - Error



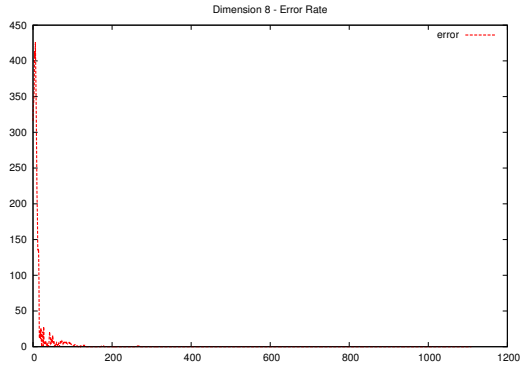
(m) *D7* - Minimum, Maximum, Average



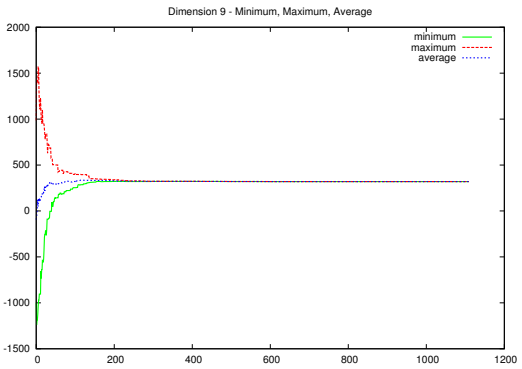
(n) *D7* - Error



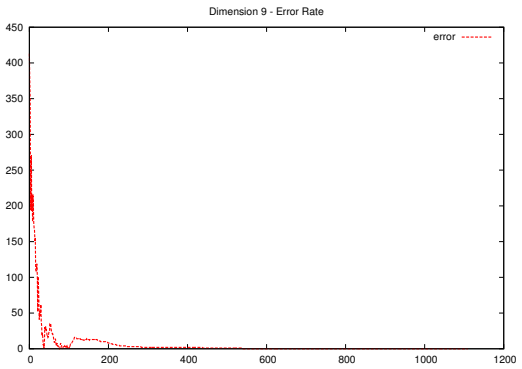
(o) *D8* - Minimum, Maximum, Average



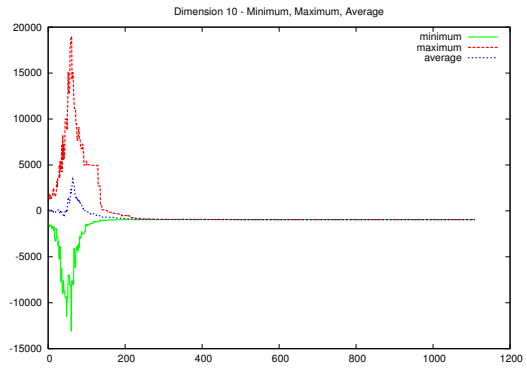
(p) *D8* - Error



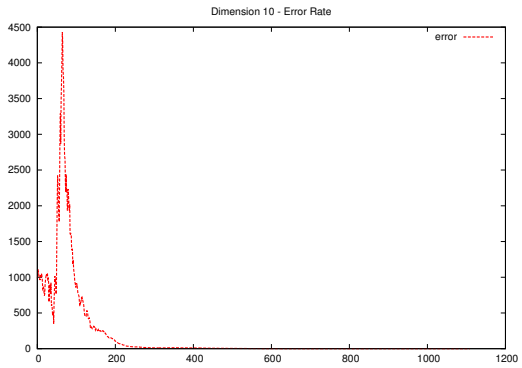
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



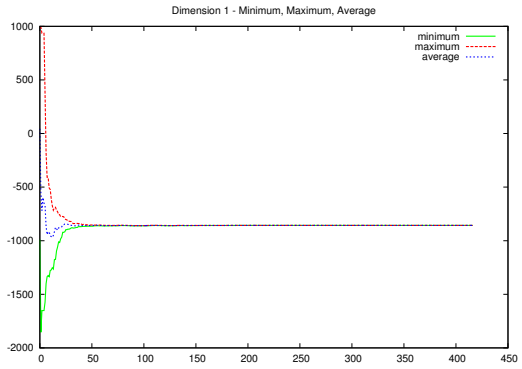
(s) $D10$ - Minimum, Maximum, Average



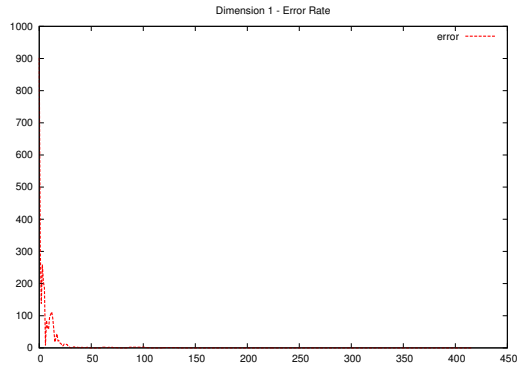
(t) $D10$ - Error

Figure A.7: Minimum, maximum, average, and error graphs produced by the DET for a typical run of a 10-D Shifted Rosenbrock's Function with control parameters $NP = 101$, $F = 0.50$ and $Cr = 0.90$.

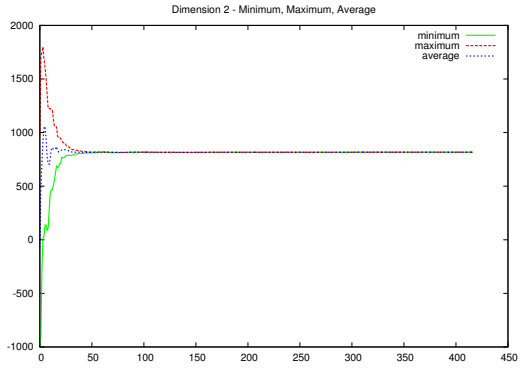
A.8 Problem F6 – DE/best-to-next/1/bin



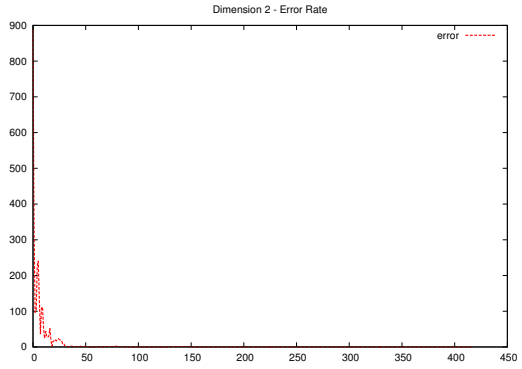
(a) $D1$ - Minimum, Maximum, Average



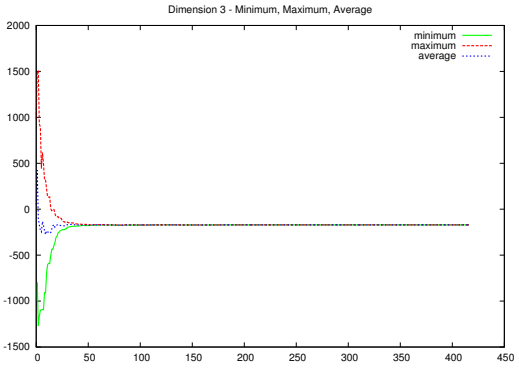
(b) $D1$ - Error



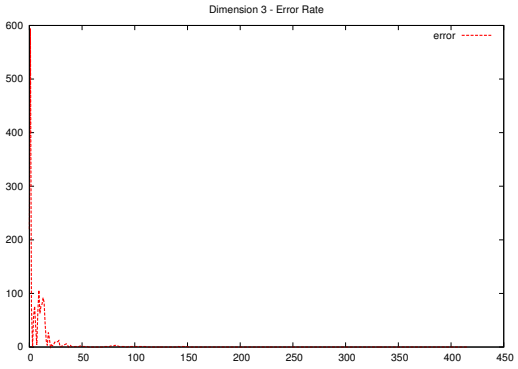
(c) $D2$ - Minimum, Maximum, Average



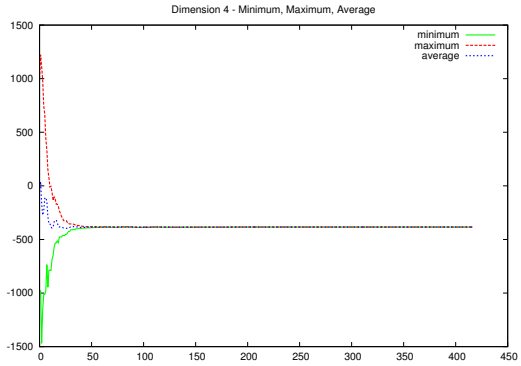
(d) $D2$ - Error



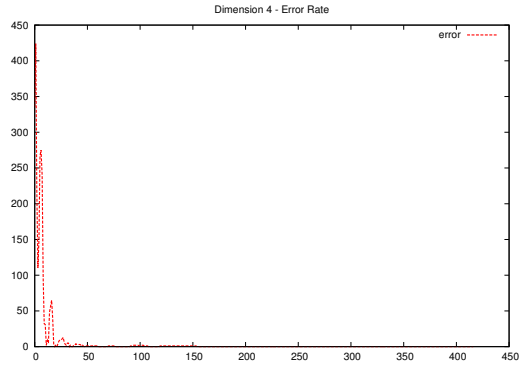
(e) $D3$ - Minimum, Maximum, Average



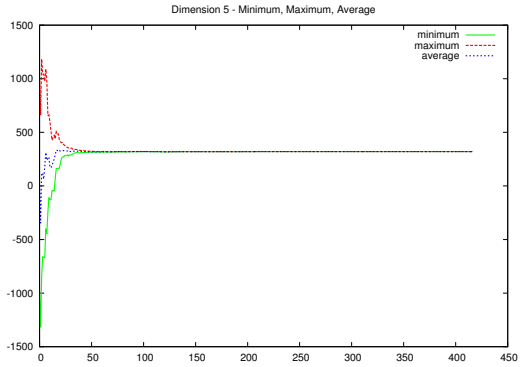
(f) $D3$ - Error



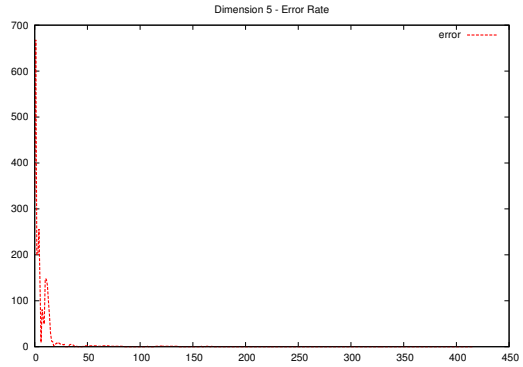
(g) $D4$ - Minimum, Maximum, Average



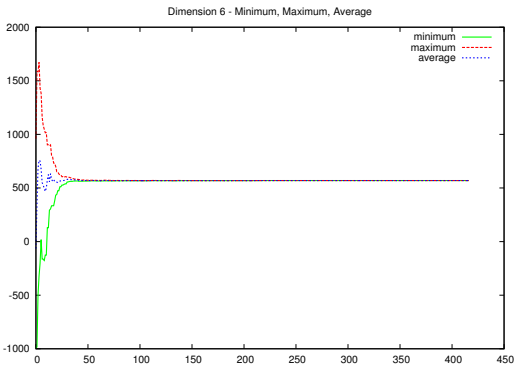
(h) $D4$ - Error



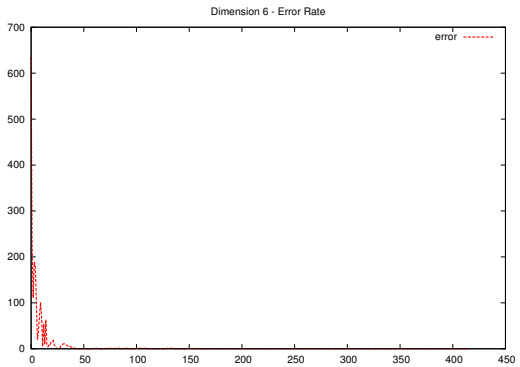
(i) $D5$ - Minimum, Maximum, Average



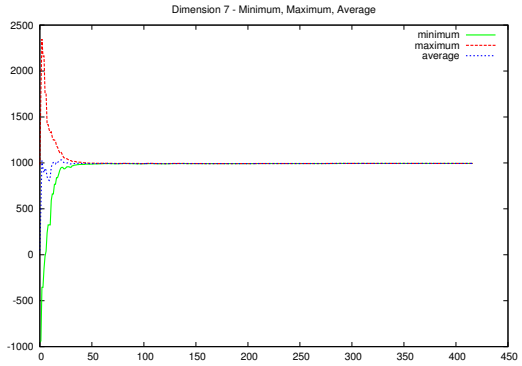
(j) $D5$ - Error



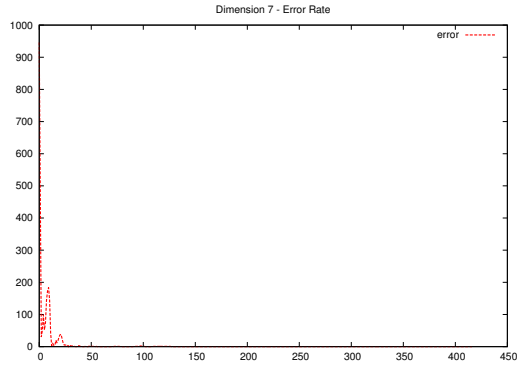
(k) $D6$ - Minimum, Maximum, Average



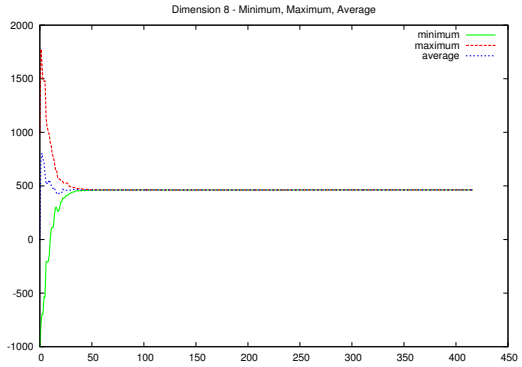
(l) $D6$ - Error



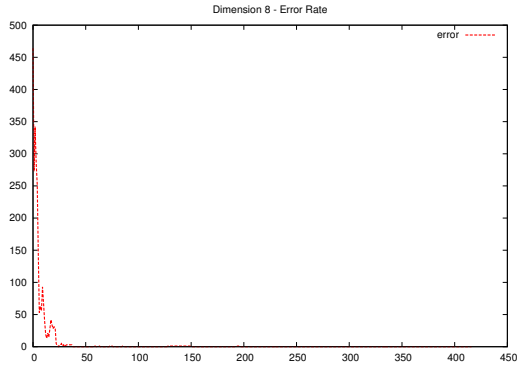
(m) *D7* - Minimum, Maximum, Average



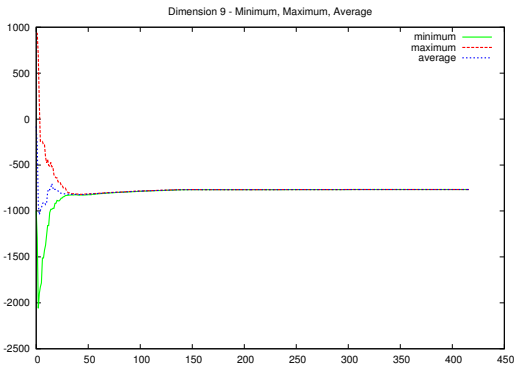
(n) *D7* - Error



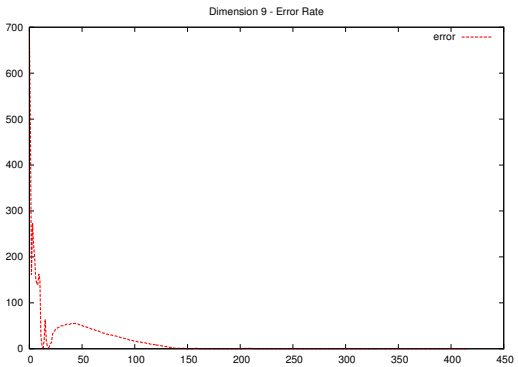
(o) *D8* - Minimum, Maximum, Average



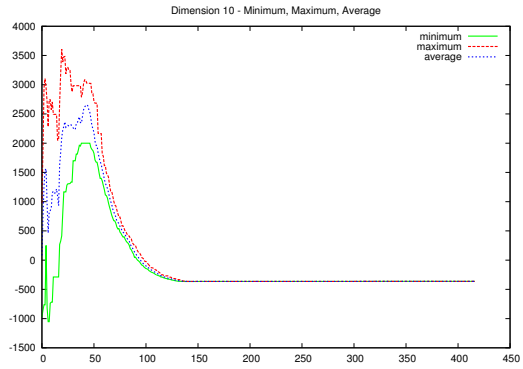
(p) *D8* - Error



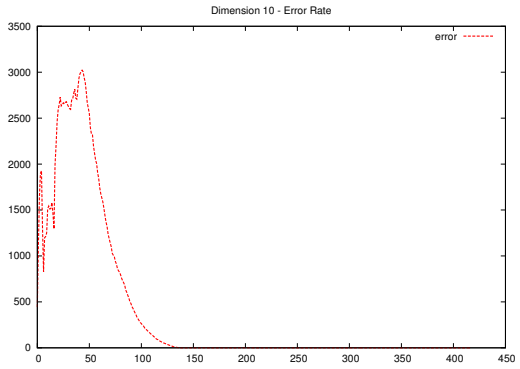
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



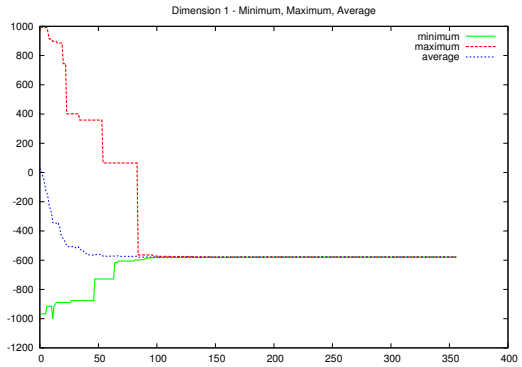
(s) $D10$ - Minimum, Maximum, Average



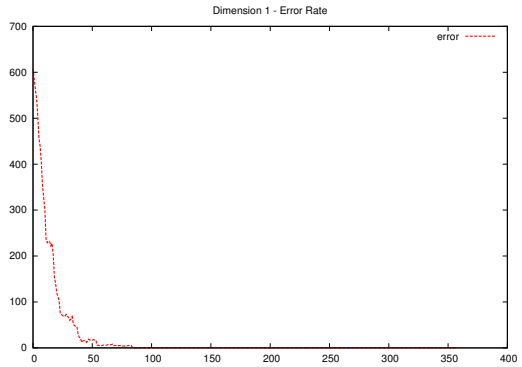
(t) $D10$ - Error

Figure A.8: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/best-to-next/1/bin* against a 10-D Shifted Rosenbrock's Function with control parameters $NP = 101$, $F = 0.60$ and $Cr = 0.90$.

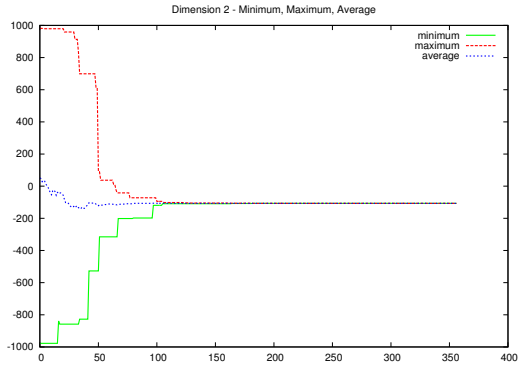
A.9 Problem F9 – DE/rand/1/bin



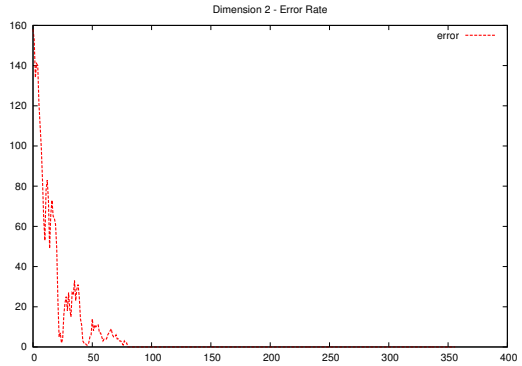
(a) $D1$ - Minimum, Maximum, Average



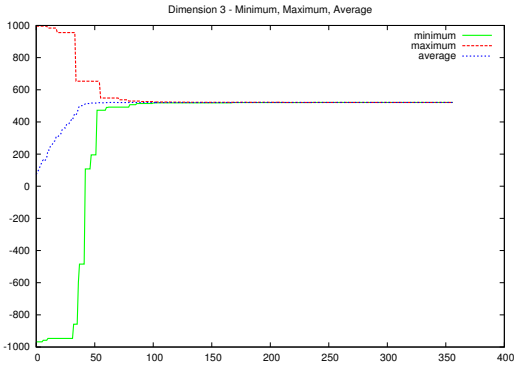
(b) $D1$ - Error



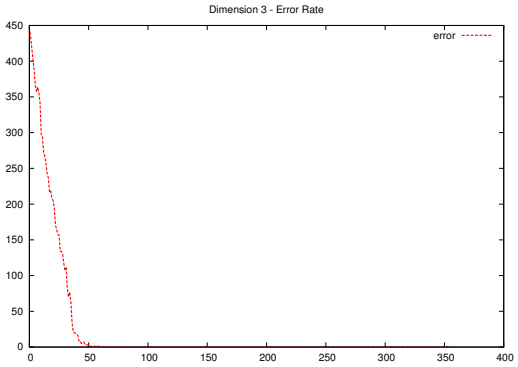
(c) $D2$ - Minimum, Maximum, Average



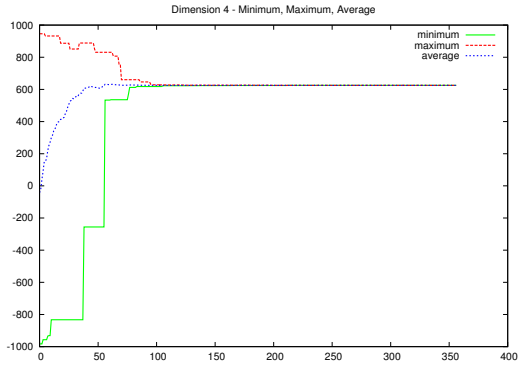
(d) $D2$ - Error



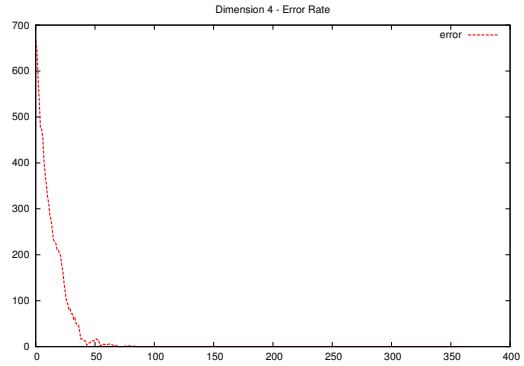
(e) $D3$ - Minimum, Maximum, Average



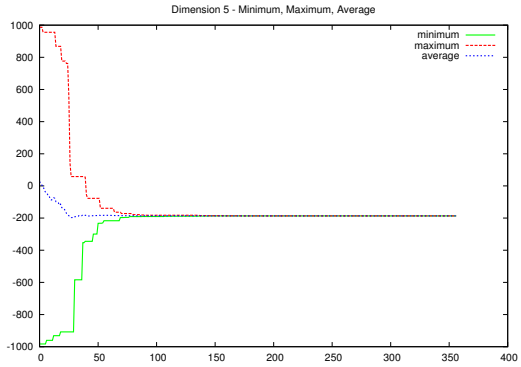
(f) $D3$ - Error



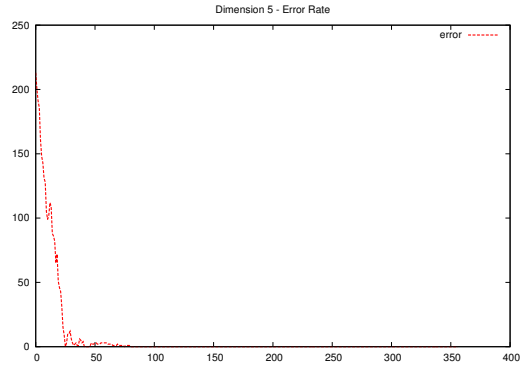
(g) $D4$ - Minimum, Maximum, Average



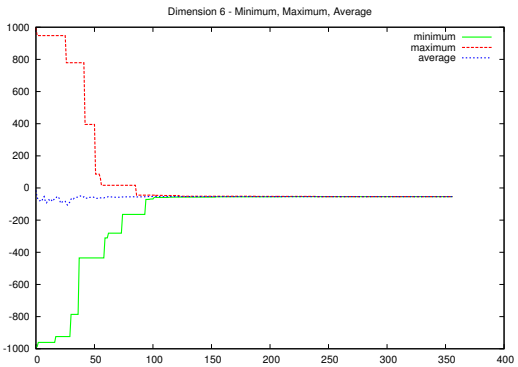
(h) $D4$ - Error



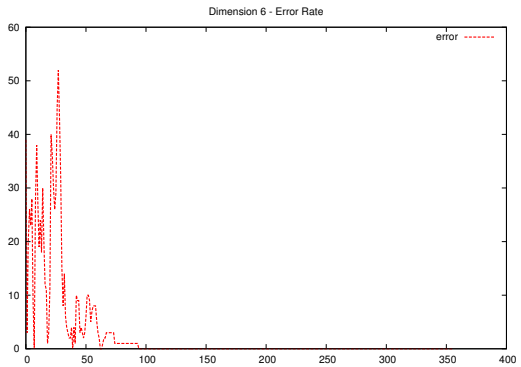
(i) $D5$ - Minimum, Maximum, Average



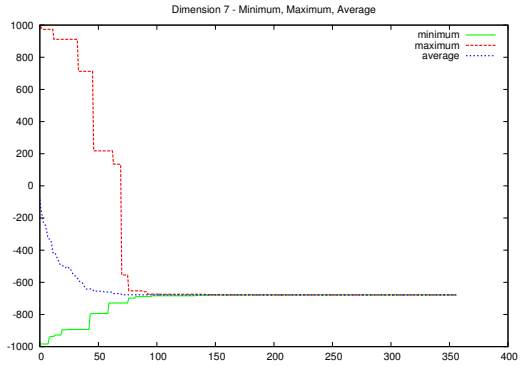
(j) $D5$ - Error



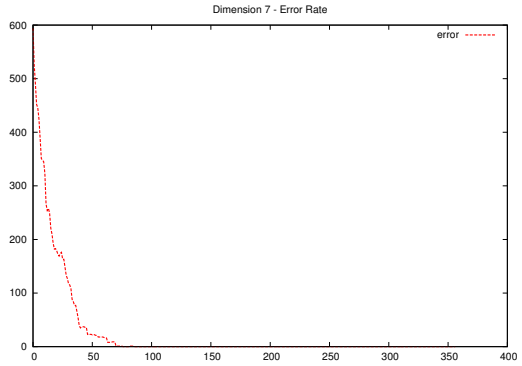
(k) $D6$ - Minimum, Maximum, Average



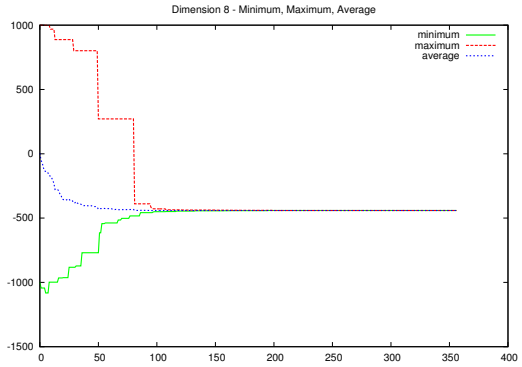
(l) $D6$ - Error



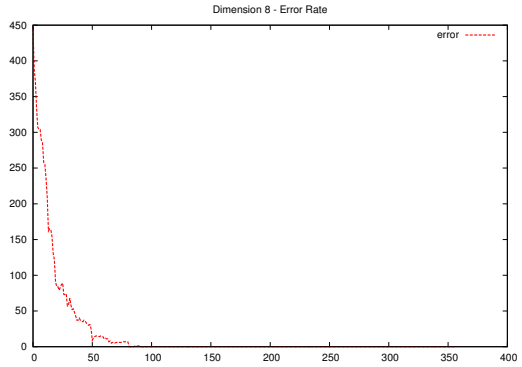
(m) *D7* - Minimum, Maximum, Average



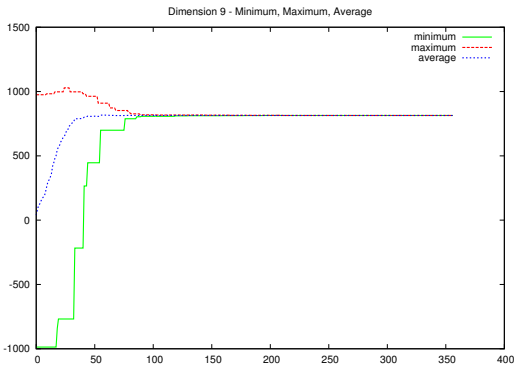
(n) *D7* - Error



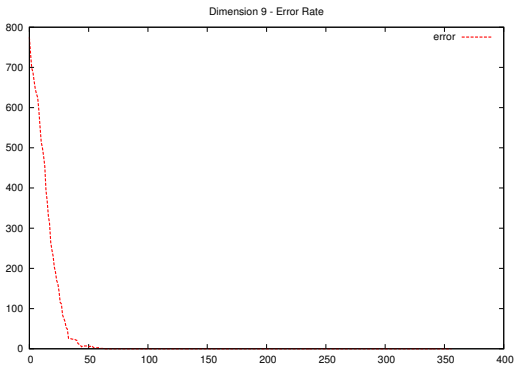
(o) *D8* - Minimum, Maximum, Average



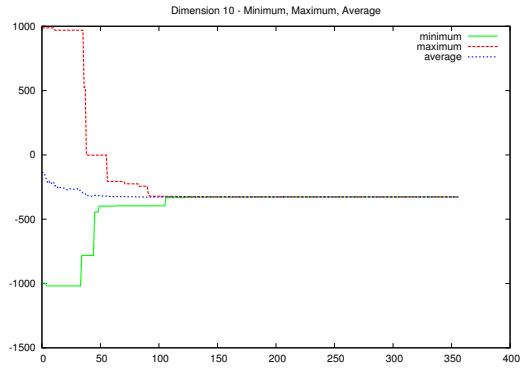
(p) *D8* - Error



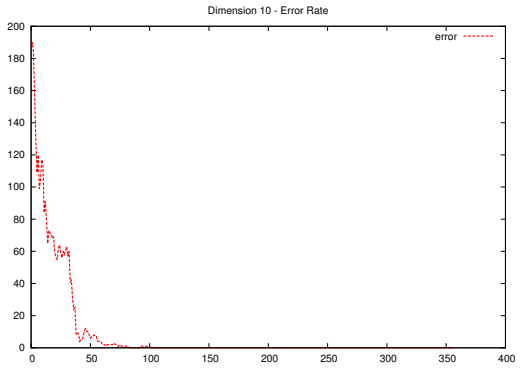
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



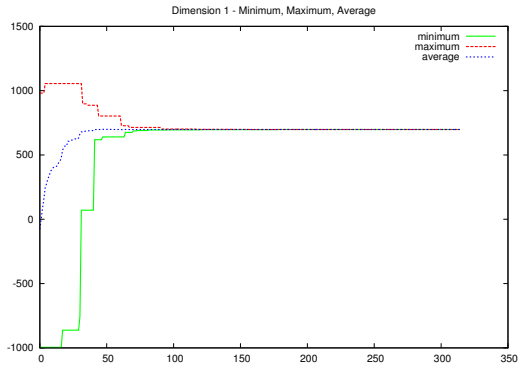
(s) $D10$ - Minimum, Maximum, Average



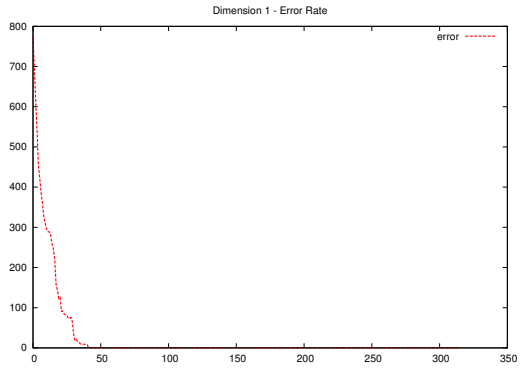
(t) $D10$ - Error

Figure A.9: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/rand/1/bin* against a 10-D Shifted Rastrigin's Function with control parameters $NP = 101$, $F = 0.10$ and $Cr = 0.00$.

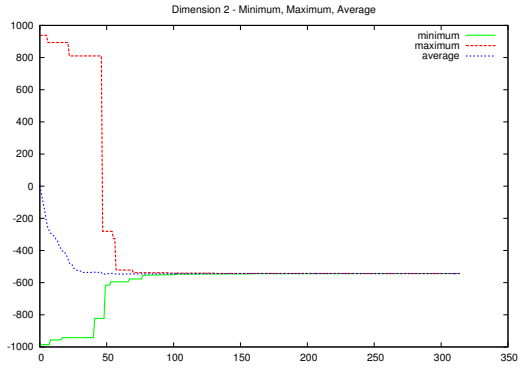
A.10 Problem F9 – DE/best-to-next/1/bin



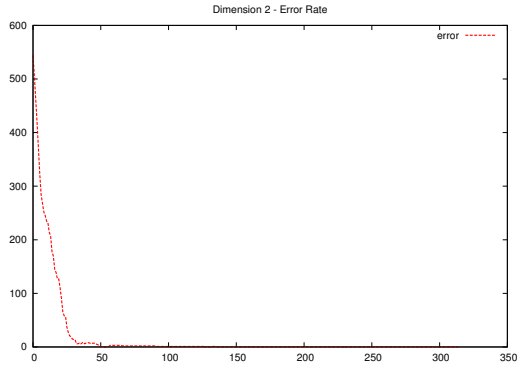
(a) $D1$ - Minimum, Maximum, Average



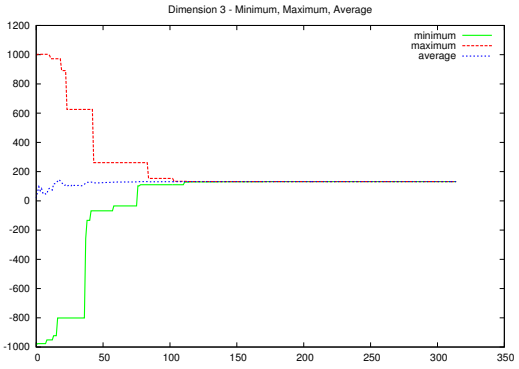
(b) $D1$ - Error



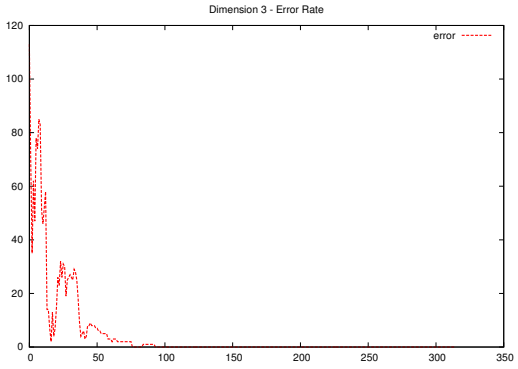
(c) $D2$ - Minimum, Maximum, Average



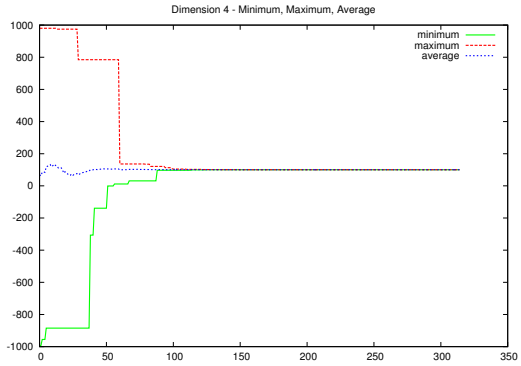
(d) $D2$ - Error



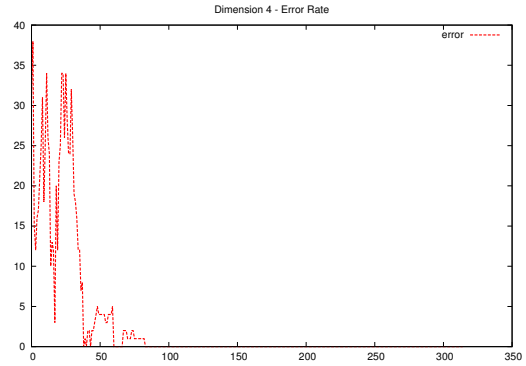
(e) $D3$ - Minimum, Maximum, Average



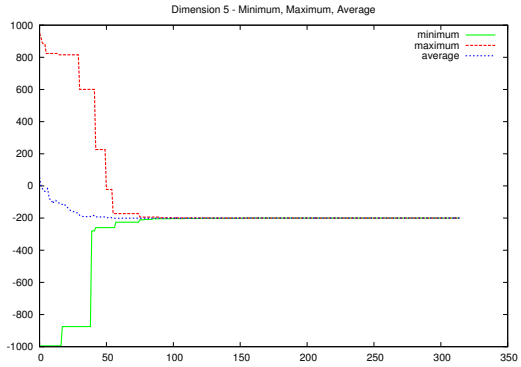
(f) $D3$ - Error



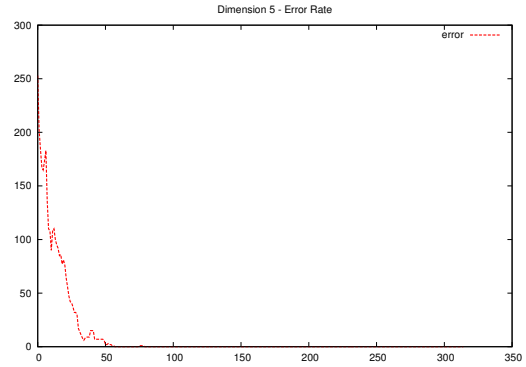
(g) $D4$ - Minimum, Maximum, Average



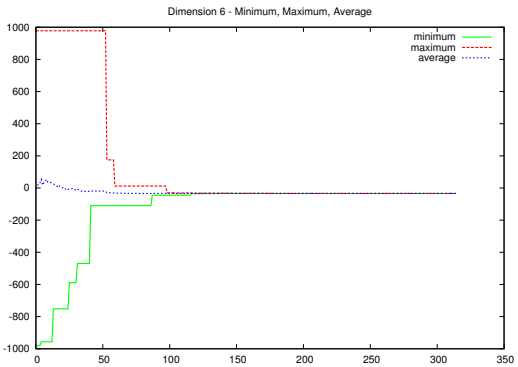
(h) $D4$ - Error



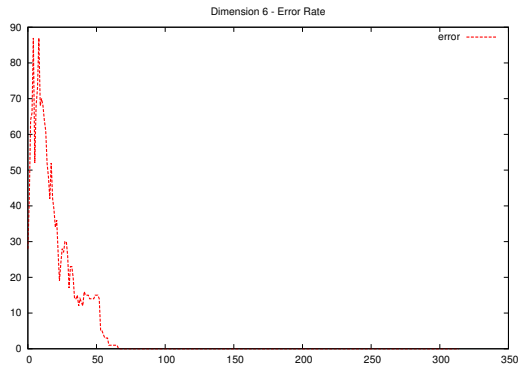
(i) $D5$ - Minimum, Maximum, Average



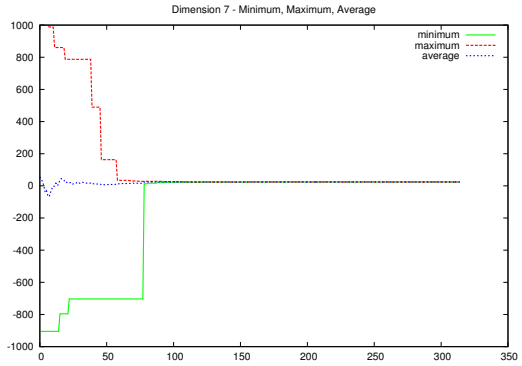
(j) $D5$ - Error



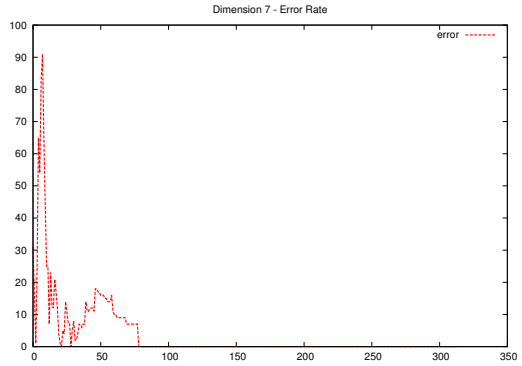
(k) $D6$ - Minimum, Maximum, Average



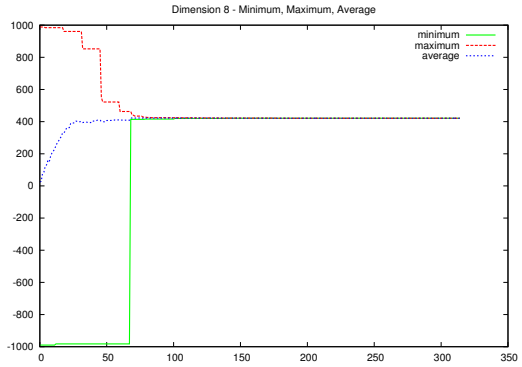
(l) $D6$ - Error



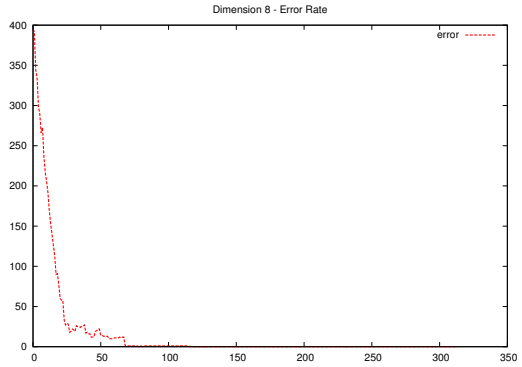
(m) *D7* - Minimum, Maximum, Average



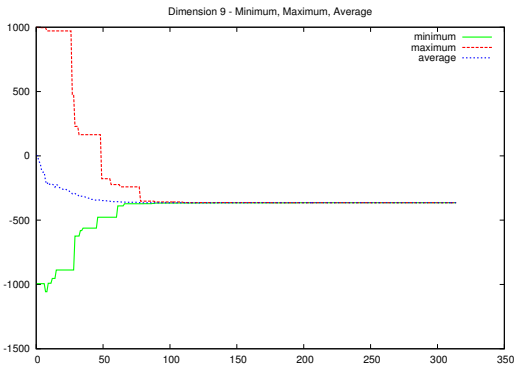
(n) *D7* - Error



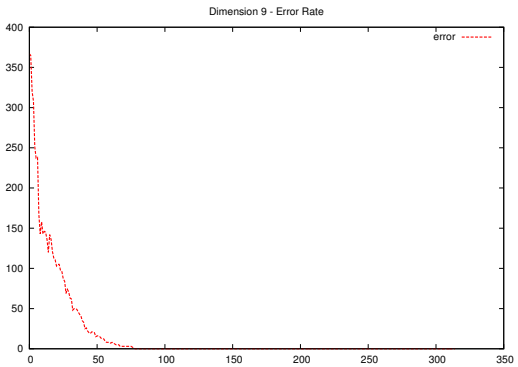
(o) *D8* - Minimum, Maximum, Average



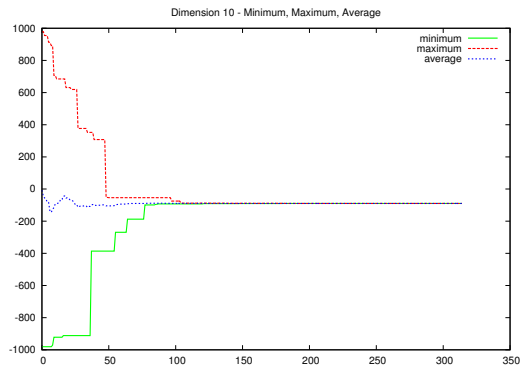
(p) *D8* - Error



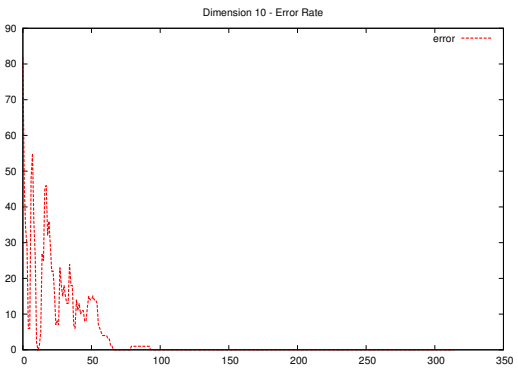
(q) *D9* - Minimum, Maximum, Average



(r) *D9* - Error



(s) $D10$ - Minimum, Maximum, Average



(t) $D10$ - Error

Figure A.10: Minimum, maximum, average, and error graphs produced by the DET for a typical run of *DE/best-to-next/1/bin* against a 10-D Shifted Rastrigin's Function with control parameters $NP = 101$, $F = 0.20$ and $Cr = 0.00$.

Appendix B

Source Code

In this section, there can be found source code excerpts that were written during the development of the Differential Evolution Toolkit (DET). Source code for each of the four main stages of Differential Evolution (DE) is provided. Additionally, the source of the objective function for problem F_6: Shifted Rosenbrock's Function is provided. Listing B.1 refers to the initialization stage, Listing B.2 to the mutation stage using the *DE/rand/1* scheme, Listing B.3 to the mutation stage using the *DE/best-to-next/1* scheme, Listing B.4 to the binomial crossover method, Listing B.5 to the selection method used for the minimization of an objective function, and, lastly, Listing B.6 contains the source code for the objective function that was written to represent F_6: Shifted Rosenbrock's Function.

```
/**
 * @brief Initialisation method.
 *
 * For things that cannot or should not be done in the class constructor.
 *
 * @return A boolean flag to tell us that initialisation was successful.
 */
bool Init() {
  // Find the range and offset from the constraints.
  T range = maxConstraint - minConstraint;
  T offset = minConstraint;
  // For each member of the population.
  for (unsigned int i = 0; i < NP; ++i) {
    // For each parameter in the problem dimension.
    for (unsigned int j = 0; j < N; ++j) {
      // Initialise the parameter in the target vector with a random number
      // in the range specified.
      targetVector[i][j] = offset + (range * (static_cast<T>(rand()) /
                           static_cast<T>(RAND_MAX)));
      // Initialise the offsets for the objective function.
      o[j] = offset + (range * (static_cast<T>(rand()) /
             static_cast<T>(RAND_MAX)));
    }
  }

  return true;
} // End of Init method.
```

Listing B.1: The initialization method source code.

```
/**
 * @brief The rand/1 mutation method.
 *
 * Performs the genetic operation called "mutation" on members of the
 * population to generate new children.
 */
void Rand1Mutation() {
  // A special vector class that acts as a shuffler and generates random,
  // mutually exclusive integers between 0 and NP.
  RandomVector<int, NP> rv;

    // For each member of the population.
    for (unsigned int count = 0; count < NP; ++count) {
      // Get our random integers from the RandomVector class.
      unsigned int i = rv.get_i();
      unsigned int a = rv.get_a();
      unsigned int b = rv.get_b();
      unsigned int c = rv.get_c();

      // For each parameter in the problem dimension.
      for (unsigned int j = 0; j < N; ++j) {
        // The mutation equation. Assign the result of the equation to the
        // donor vector.
        donorVector[i][j] = targetVector[a][j] + kScaleFactor *
                            (targetVector[b][j] - targetVector[c][j]);
      }
    }

    // We are done with this dimension. Shuffle the vector and proceed to
    // the next dimension.
    rv.shuffle();
} // End of Mutation method.
```

Listing B.2: Source code for the *DE/rand/1* mutation scheme.

```
/**
 * @brief The best-to-next/1 method.
 *
 * Performs the genetic operation called "mutation" on members of the
 * population to generate new children.
 */
void BestToNext1Mutation() {
  // Our parent vector is sorted by fitness - the best is at index zero.
  unsigned int best = 0;
  unsigned int n1;
  unsigned int n2;
```

```
12
13    // For each member of the population, excluding the last one.
14    for (unsigned int i = 0; i < (NP - 1); ++i) {
15      // n1 begins at index one and ends at index (NP - 1).
16      n1 = i + 1;
17      // n2 begins at (NP - 1) and ends at index one.
18      n2 = NP - n1;
19
20      // For each parameter in the problem dimension.
21      for (unsigned int j = 0; j < N; ++j) {
22        // The mutation equation. Assign the result of the equation to the
23        // donor vector.
24        donorVector[i][j] = targetVector[best][j] + kScaleFactor *
25                            (targetVector[n1][j] - targetVector[n2][j]);
26      }
27    }
28
29    // Copy the best member of the population to the last index of the donor
30    // vector, forcing it to re-combine with the worst member of the target
31    // vector during the crossover stage.
32    for (unsigned int j = 0; j < N; ++j) {
33      donorVector[NP-1][j] = targetVector[best][j];
34    }
35  }
```

Listing B.3: Source code for the *DE/best-to-next/1* mutation scheme.

```
1  /**
2   * @brief The binomial crossover method.
3   *
4   * Performs the genetic operation called "crossover" on members of the
5   * population to generate new children. This uses the binomial crossover
6   * method.
7   */
8  void BinomialCrossover() {
9    // For each member of the population.
10   for (unsigned int i = 0; i < NP; ++i) {
11     /* For every member of the population, generate a random number between
12        0 and N. This will be used to guarantee that at least one dimension
13        of the donor vector is passed to the trial vector. */
14     unsigned int n = static_cast<int>(rand()) % N;
15
16     // For each parameter in the problem dimension.
17     for (unsigned int j = 0; j < N; ++j) {
18       // A random number between 0 and 1.
19       double crossoverTest = static_cast<double>(rand()) /
20                              static_cast<double>(RAND_MAX);
```

```
21
22      /* If crossoverTest is less than or equal to the crossover rate, the
23         trial vector is populated with the donor vector's value.
24         Otherwise, it is populated with the target vector's value.
25         To guarantee that at least one dimension of the trial vector is
26         populated with one from donor vector, n is used. n will always
27         populate the trial vector, regardless of the crossover test. */
28      if ((crossoverTest <= kCrossoverRate) || (j == n)) {
29        trialVector[i][j] = donorVector[i][j];
30      } else {
31        trialVector[i][j] = targetVector[i][j];
32      }
33    }
34  } // End of outer for loop.
35} // End of BinomialCrossover method.
```

Listing B.4: Source code for the binomial crossover method.

```
1 /**
2  * @brief The selection method.
3  *
4  * Performs selection on the members of the population. The "fittest" -
5  * the ones that bring us closer to the result - will survive and be passed
6  * on to the next generation. This method is for minimisation.
7  *
8  * param[in] pFunc A pointer to our objective function.
9  */
10 void MinimiseSelection(T (*pFunc)(T[], T[])) {
11   /** An array of size N to hold the values of all dimensions of the target
12    *  vector at the same index.
13    */
14   T tempTargetVector[N];
15
16   /** An array of size N to hold the values of all dimensions of the trial
17    *  vector at the same index.
18    */
19   T tempTrialVector[N];
20
21   // A flag that will tell us if the trial vector at index i provides a
22   // better solution than its corresponding target vector.
23   bool isABetterSolution = false;
24
25   // For each member of the population.
26   for (unsigned int i = 0; i < NP; ++i) {
27     // For each parameter in the problem dimension.
28     for (unsigned int j = 0; j < N; ++j) {
29       // Copy all dimensions of the target vector and trial vector in
```

```
30        // arrays.
31        tempTargetVector[j] = targetVector[i][j];
32        tempTrialVector[j] = trialVector[i][j];
33      }
34
35      // If the trial vector provides a better solution than the target
36      // vector, the flag is true.
37      if (pFunc(tempTrialVector, o) <= pFunc(tempTargetVector, o))
38        isABetterSolution = true;
39
40      /* If we have a better solution, we copy all dimensions of the ith
41         member of the trial vector population to the ith index of the target
42         vector population, otherwise the target vector keeps the same values
43         for the next generation. */
44      if (isABetterSolution) {
45        for (unsigned int k = 0; k < N; ++k) {
46          targetVector[i][k] = trialVector[i][k];
47        }
48      }
49
50    // Reset our flag to false.
51    isABetterSolution = false;
52
53    } // End of outer for loop.
54  } // End of MinimiseSelection method.
```

Listing B.5: Minimization of an objective function source code.

```
1  /**
2   * @brief F6: Shifted Rosenbrock's Function.
3   *
4   * @param[in] pD The candidate solution.
5   * @param[in] o  The offsets of the objective function.
6   *
7   * Properties: Multimodal, Non-separable.
8   * Global Optimum = f_bias
9   */
10 template<typename T, unsigned int N>
11 T ShiftedRosenbrocksFunction(T pD[], T o[]) {
12   T result = 0.0;
13   T f_bias = 0.0;
14
15   T temp1, temp2, temp3;
16   for (unsigned int i = 0; i < (N - 1); ++i) {
17     // First thing we do is make ( z_i^2 - z_(i+1) )
18     // Where: z = x - o + 1
```

```
19      temp1 = (((pD[i] - o[i] + 1) * (pD[i] - o[i] + 1)) - (pD[i+1] - o[i+1] +
            1));
20      // Next we square and multiply by 100
21      temp2 = 100.0 * temp1 * temp1;
22      // Next we make a component from ( z_i - 1 )^2
23      // The compiler will remove the +1 -1 at any level above -O0
24      temp3 = (pD[i] - o[i] + 1 - 1) * (pD[i] - o[i] + 1 - 1);
25
26      // Now we form the loop accumulation.
27      result += temp2 + temp3;
28    }
29
30    result += f_bias;
31
32    return result;
33  }
```

Listing B.6: F_6: Shifted Rosenbrock's Function source code.