2015

# Central force optimization : analysis of data structures & multiplicity factor

Matthew Bick
*University of Toledo*

A Thesis

entitled

# Central Force Optimization - Analysis Of Data Structures & Multiplicity Factor

by

Matthew Bick

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Masters of Science Degree in Engineering

Dr. Robert C. Green II, Committee Chair

Dr. Mansoor Alam, Committee Member

Dr. Henry Ledgard, Committee Member

Dr. Ezzatollah Salari, Committee Member

Dr. Patricia Komuniecki, Dean
College of Graduate Studies

The University of Toledo
December 2015

An Abstract of

Central Force Optimization - Analysis Of Data Structures & Multiplicity Factor

by

Matthew Bick

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Masters of Science Degree in Engineering

The University of Toledo
December 2015

Central Force Optimization is a relatively new (2007) optimization technique based on Newtonian equations for objects moving through friction-less space. Unlike Particle Swarm Optimization, which is in the same family, CFO is deterministic (it does not contain elements of randomness). This paper explores two separate items as they relate to the CFO algorithm. First, tests indicate differences in runtime as a result of underlying data structure (disjoint arrays vs object encapsulation). It is shown that an implementation with data placed in disjoint arrays is consistently found to run faster than an implementation where data is encapsulated within an object. Second, tests measure the runtime and convergence effects of a *multiplicity factor*. When a multiplicity factor is employed, runtimes are significantly faster for systems of probes which converge before the maximum limit is reached. These systems also tend to converge in less iterations than the CFO in its standard form.

This work is dedicated to my family.

# Acknowledgments

Thank you first and foremost to my adviser, Dr. Robert Green. Without your advice and direction, this work would not have been possible.

Special thanks to Kevin Borrowman for years of inspiration and encouragement. Thank you for opening my eyes to a world of knowledge and a new way to interact with the world around me.

# Contents

# List of Tables

# List of Figures

# List of Abbreviations

| | |
|---|---|
| CFO | Central Force Optimization |
| dim | Dimension |
| hyper | Hypercube |
| IDP | Initial Distribution Pattern |
| ks test | Kolmogorov-Smirnov Test |
| nP | Number of Probes |
| OO | Object-Oriented |
| pD | Probe Density |
| UOA | Uniform On Axis |
| UOD | Uniform On Diagonal |

# List of Symbols

| | |
|---|---|
| $P$ ..... | probe |
| ${}_iP$ ..... | probe with identifier $i$ |
| ${}_iP_{m_t}$ ..... | probe $i$'s mass at time $t$ |
| ${}_iP_{f_t}$ ..... | probe $i$'s multiplicity factor at time $t$ |
| ${}_iP_{\vec{a}_t}$ ..... | probe $i$'s acceleration vector at time $t$ |
| ${}_iP_{\vec{p}_t}$ ..... | probe $i$'s positional vector at time $t$ |
| ${}_iP_{a_{d_t}}$ ..... | probe $i$'s acceleration in dimension $d$ at time $t$ |
| ${}_iP_{p_{d_t}}$ ..... | probe $i$'s position in dimension $d$ at time $t$ |

# Chapter 1

# Introduction

The computational power of computers brought about new ways to approach problems. Previously, solution-searching techniques which required hundreds or thousands of calculations would not have been considered a viable option. Therefore, optimization techniques, as they exist today, which often require an enormous amount of operations, were inconceivable. Optimization problems are those which search for solutions within a given search-space (with an unknown topology), constrained by certain conditions. The best solution, however, may not always be attainable due to additional constraints such as runtime. Even with the current technology, there exist problems in which a perfect solution may require lifetimes to calculate.

Current optimization techniques require a problem to be encoded in the form of a symbolic equation (an *objective function*) before they can be used. The equation may be a mathematical function, a sequence of movements, or any other sufficient way of representing a problem. This allows each solution-candidate to be ranked based on its closeness, or *fitness*, to the objective function. Much effort has been, and is still required to be, put into research toward the enhancement of current techniques for use with problems of greater complexity.

Though new optimization techniques appear each year, there exist well-established approaches which have proven to work well for certain types of problems. Many new techniques are derivative of one of these established techniques. To better understand the Central Force Optimization algorithm, these techniques will be briefly described in the following chapter. These established algorithms are often derived from the physical world and, thus, are best described by analogy to their real-world counterparts.

## 1.1 Structure

This thesis describes the background information and details involving two separate contributions to the CFO algorithm. The *Background* chapter contains information about the CFO analogy and its structure. The *Python Objects in CFO* chapter describes how runtimes are affected when an array-based data structure is replaced by a structure in which the data is encapsulated within a computational object. The *Multiplicity Factor* chapter details differences in runtimes when a *multiplicity factor* is implemented in addition to the original (array-based) algorithm. *Conclusions & Future Work* informs the reader about what can be concluded from this work and outlines areas which may be of interest for further research. Finally, an appendix includes supplementary material including test configurations, initial probe distribution patterns, statistical results tables, and other resources.

# Chapter 2

# Background

CFO is one of a handful of established optimization techniques, each technique possesses its own unique way of searching for solutions in a defined space. Certain techniques work better for some types of problems than for others. The major optimization techniques are described here to provide the reader with a better understanding of how CFO fits into this world of optimization algorithms.

## 2.1 Major Optimization Techniques

There are many common optimization techniques. Some of the most popular, and direct competitors with CFO, are Ant Colony Optimization, the Genetic Algorithm, Genetic Programming, Particle Swarm Optimization [1].

Ant Colony Optimization (ACO) uses probes which mimic the seemingly random movements of a colony of ants to explore a given area in search of food (optimal solutions) [2] [3]. Just as ants deviate from their path in the physical world, probes randomly deviate from their path in order to explore nearby solutions. This continues until a termination criteria is reached (such as a pre-established amount of time has elapsed or a path is found which is considered sufficient within predefined parameters). This type of algorithm tends to be very useful when applied to shortest-path problems

or problems where multiple paths may be required.

The Genetic Algorithm (GA) is modeled after DNA and utilizes a "survival of the fittest" approach [4] [5]. A population of *chromosomes* is used to to represent possible solutions. A chromosome may be represented by an array of numbers, letters, or other symbolic elements which represent possible solutions for the objective function. For instance, a chromosome may represent a path through a maze as an array of directions, such as "up, down, down, left, right, ...", or numeric coefficients to a mathematical objective function. The chromosome then "breeds" (combines elements in a predefined method) to form succeeding populations. Breeding is based on randomness, but gives priority to solutions of greater fitness. Generally, a newly-formed "child" chromosome will copy elements from each of its "parent" chromosomes. In this way, parent chromosomes live on through their "children". Of course, since this is not a physical system, there are options available which nature does not contain. For instance, "immortality" may be granted to a given number of chromosomes from a population by copying it directly to the succeeding population. In this way, optimal solutions are not lost in the progression of generations; this ensures that, if the optimal solution is found before termination criteria is met, it is not modified. "Mutations" also occur in this procedure as a given percentage (usually very low, 0.01 % or $\frac{1}{10000}$) of chromosomes have a predefined number of elements randomly modified. This algorithm is useful for shortest-path problems, problems involving binary solutions, and problems which benefit from searching a great variety of diverse solutions [6].

Genetic Programming (GP) is unique in that it searches for an equation to represent a set of data points as well as the best fit coefficients which fit the equation (Fig. 2.1) [7] [8] [9]. This algorithm exploits the properties of a data tree to find solutions. Generally, each node in the tree represents an operator (addition, multiplication, sine, cosine, etc) and the branches represent the relation of terms. The final nodes

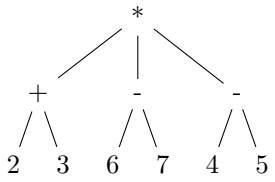


Figure 2-1: GP Tree Representing $(2+3)*(6-7)*(4-5)$

(or leaves) on each branch hold the numeric coefficients which will be inserted into each operator.

GP solutions are "bred" using the same principles established for GA, therefore, a chromosome may be developed which contains information to fill the tree. Unlike GA, GP chromosomes carry information for coefficients as well as operators; modifications must be made to the GA in order to account for the operators as part of the breeding operation.

Particle Swarm Optimization (PSO) uses probes which represent satellites in frictionless space [10] [11]. The fitness of the particle's solution relates directly to the particle's mass. Each particle is given a random initial velocity and starting position. The particle then moves according to Newtonian equations using three factors which are assigned to the particle. These are factors act as the particle's velocity and two separate acceleration forces. The particle's first acceleration factor is based on its attraction to the particle with the greatest mass (of the entire system). The second acceleration factor is based on its attraction to the particle with the greatest mass within a predetermined neighborhood of its position. There may also be a random coefficient involved with each of these factors which helps the system of probes to avoid cycles. This algorithm is generally suitable for problems which require great precision, but may suffer when the search-space is greatly dispersed.

Central Force Optimization (CFO), the algorithm of interest for this paper, is in the same family as PSO, thus there are many similarities among the two algorithms. Both techniques explore the search-space using multiple probes which move in accordance with the Newtonian equations for velocity and acceleration of physical bodies in friction-less space. These algorithms also use probes which are analogous to satellites (rather than chromosomes) to represent possible solutions (*solution candidates*). In each, the dimensional coordinates of a probe represent a possible solution to the objective function. Probes are assigned an initial position and an initial velocity, the probe's mass is determined by its representative solution's fitness to the objective function. The probes move in discrete time intervals based on attractions to each other based on calculations using Newton's gravitational equation.

Though CFO and PSO share a common basis, there are some very important differences. First, while PSO's probes are attracted to the a) local-best solution and b) global-best solution, CFO's probes are attracted to every other probe which has greater or equal mass. Therefore, the probe of least mass is influenced by n-1 probes and the probe of greatest mass is not influenced by any other probes (assuming no probe has equal mass). Second, PSO's probes are initially distributed throughout the search-space an a random fashion, CFO's probes are distributed using one of several patterns (covered in detail in the following sections). Since probes model physical bodies in space with known equations, their paths can be reproduced and there is no concept of random change after the CFO algorithm begins running. In contrast, PSO involves random coefficients for acceleration and velocity which make reproduction impossible. In CFO, this can be useful in determining how solutions relate to each other.

Though they may appear subtle, these differences are much more important than they may seem. Of these algorithms, CFO is the only technique which does not

include random events, that is, it is *deterministic*. Therefore, it requires only a single run to achieve its solution, subsequent runs will produce the exact same results. Non-deterministic algorithms (those with elements of randomness) require multiple runs to ensure an appropriate solution has been reached. Since non-deterministic algorithms use randomly initialized solution candidates, there is a possibility that the solutions could be placed tight group and become constrained to a locally-optimal solution. Non-deterministic algorithms require multiple runs to ensure that the initial probe placements produce adequate coverage of the search-space. Hence, when considering runtimes between deterministic and non-deterministic algorithms, one must remember that non-deterministic runtimes must be multiplied by a number which is consistent with the amount of runs necessary to feel that adequate coverage has been achieved.

## 2.2 Central Force Optimization

Central Force Optimization was developed by Richard A. Formato in 2007 [12] [13] [14] [15] [16] and was proven to converge in 2010 [17] [18]. It has since been applied to a diverse set of problems ranging from Electromagnetics [19] [20] & Antenna Design [21] [22] [23] [24] [25] to the detection of water leakage [26] and Biogeography [27]. Benchmark tests have shown that CFO is a very capable algorithm [28]. CFO has can also be used to train other optimization techniques, such as neural networks [29]. Most recently, CFO has been used for path planning for the three-dimensional unmanned aerial vehicles (UAV) [30].

### 2.2.1 Advancements

CFO has undergone a series of advancements since its inception. It has been modified to include an automatically adjusting (shrinking) search space [31]. A parameter-

free version has also been developed which only requires the input of an objective function [32]. Many versions (including the versions in this work) involve variable initial probe positions [33]. CFO has been implemented on a GPU in an attempt to decrease computational time while exhibiting superior solution quality [34] [35] [36]. A hybrid version of CFO (CSM-CFO), based on the Simplex Method, Clustering Technique, and CFO, presents an excellent trade-off between the exploitation of the Simplex Method and the exploration of CFO [37] [38]. Most recently, an adaptive form of CFO has been developed which is based on the stability theory of discrete time-varying dynamic systems [39].

## 2.3 CFO Advantages

As previously stated, CFO is completely deterministic, which is a great advantage over non-deterministic algorithms when considering probe movements can be traced through discrete time intervals. This allows one to discover trends in probe movement. Once a trend is discovered, the search-space may be shrunk to the area of interest and the algorithm re-run to gain greater precision. While the idea of randomness may allow algorithms to find solutions in obscure areas of a search-space, the deterministic nature of CFO enables greater precision, once probes have detected the general vicinity of a suitable solution.

## 2.4 CFO Disadvantages

Since a probe's movement relies on the properties of other probes, there are a great number of calculations required to determine the new position for each probe. This gives the algorithm to exhibit a worst-case complexity of $O(n^2)$ [29] and is the inspiration for the enhancements discussed in this work. Additionally, the equations

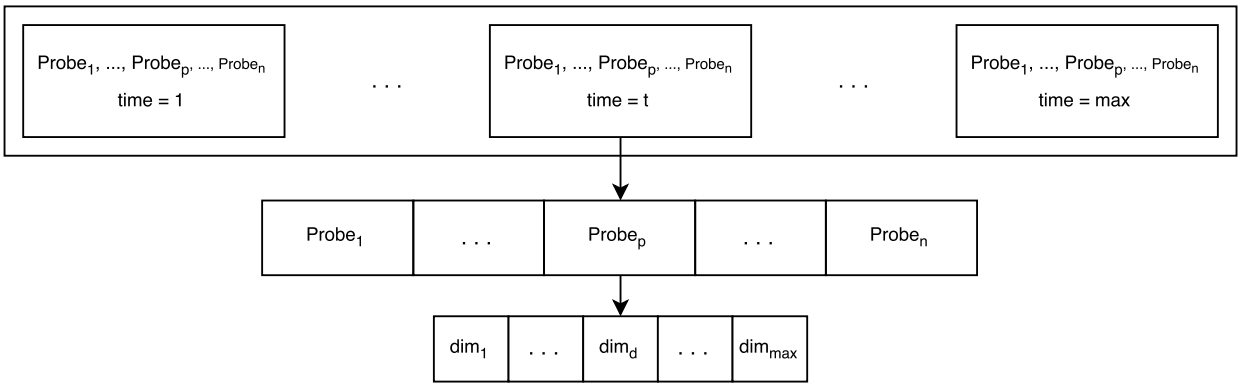


Figure 2-2: Structure of a *property array*

must accurately calculate a large number of variables for them to work properly. This can create difficulty when implementing certain parts of the CFO algorithm. This is in contrast to an algorithm such as GA, which is relatively easy to implement each piece, once the basic structure is realized.

## 2.5 CFO Structural Components

CFO implementation requires three *property arrays* (*Acceleration*, *Mass*, and *Position*) for storing data. Each property array requires a single element for each probe, probe information is consistent across arrays by index. The $i^{th}$ index in each array accounts for acceleration, mass, and positional information pertaining to probe $i$. Additionally, an parent array must be created for each property array which contains an element for each time step (Fig. 2-2). The time step elements in this parent array will represent each property at a given time. For instance, the parent array for Acceleration will contain $T$ time step elements. For a given time step element $t$, this array will contain the acceleration data for each probe (Fig. 2-3).

```
def createStructures(probeDist):
    .
    .
    .
    Accel     = [ [ [ 0 for d in range(dim) ] for p in range(numProbes) ] for t in range(max_time)]
    Mass      = [ [ 0 for p in range(numProbes) ] for t in range(max_time) ]
    Position  = [ [ [ 0 for d in range(dim) ] for p in range(numProbes) ] for t in range(max_time)]
    .
    .
    .
```

Figure 2-3: Array Structure Initialization

For the work contained in this paper, initial velocity has been omitted (set to a zero value) from the original algorithm. This change urges the probes to converge by grouping near a solution rather than forming orbits around a given solution. It is difficult to detect whether the system of probes has converged if orbits exist. Additionally, the orbits carry the possibility of keeping such a distance from a current-best candidate-solution that the space between the orbit and the central solution (if one exists) would never be adequately explored.

## 2.6 Algorithm & Implementation Details

In the following sections, unless explicitly described, it should be assumed that the CFO implementation used for testing adheres to the original algorithm, as described by Formato [19], with the exception of initial velocity, as previously stated. Any deviation from the original algorithm will be detailed as necessary. The version of the algorithm described here will be referred to as the *Array-based* and *Standard* algorithms when describing results compared to the *Object* and *Multiplicity* forms of the algorithm, respectively.

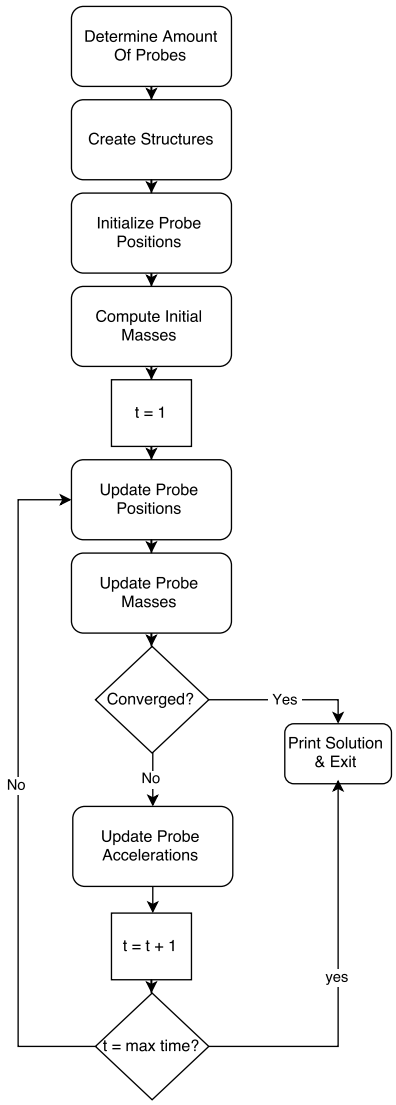


Figure 2-4: Algorithm Flow Chart

```
determine amount of probes
create structures
initialize probe positions
compute initial masses
compute initial accelerations
for time = 1 until time = maximum do
    update probe positions
    update probe masses
    if probes have converged then
        break
    end if
    update probe accelerations
end for
```

Algorithm 1: CFO Algorithm Pseudocode

## Create Probes

A probe is a computational concept which references relevant information about a particular solution-candidate. A probe is analogous to a satellite which exists in friction-less n-space, where n is the number of input parameters for the objective function. As satellites are described in terms of their name and properties (acceleration, mass, velocity, etc.), probes contain information about a solution-candidate's properties (acceleration, mass, and position) an identity in the form of numeric index (position within an array). The positional coordinates of a probe represent the inputs to the objective function. When a probe is created, it is assigned acceleration and mass values of zero. Since the initial velocity is being ignored, that value is also set to zero.

## Initial Probe Distributions

The amount of probes and the initial position of each probe varies as one of three user-selected options, *uniform-on-axis*, *uniform-on-diagonal*, or *hypercube* [40] [41]. Each distribution requires a different amount of probes due to a differing complexity

in each pattern. Each distribution generally displays an adequacy for covering the search space due to *overshoot*. This happens when a probe is moving toward another with such speed that it is able to surpass the probe which is exhibiting the influence. In this way, probes are able to explore possible solutions in areas around an ideal solution while they converge. There is some concern that probes can form periodic patterns around a central solution due to the arrangement of the group of probes. For this reason, a maximum acceleration has been implemented (further discussed in the *Updating A Probe's Acceleration* section, later in this chapter).

*Uniform-on-axis*, *uniform-on-diagonal*, and *hypercube* require differing amounts of probes. In each of the following, *dim* is the dimension of the objective function input parameters and *density* is the user-determined (natural number) amount which describes how closely the probes should be arranged to each other within the search space. Typically, the *density* setting represents how many probes should be placed per axis. In the following figures the parameters are set at $-5 \leq x \leq 2$, $-3 \leq y \leq 4$, $density = 5$, and $dim = 2$.

The *uniform-on-axis* (Fig. 2-5) distribution aligns probes to each dimensional axis in the search space. A probe is placed at the minimum and maximum boundary point for each dimension. For each axis, the remaining $density - 2$ probes will be evenly spaced between the boundary probes on each axis. The amount of probes required for this distribution can be determined by multiplying the *dim* and *density* parameters.

The *uniform-on-diagonal* (Fig. 2-6) distribution aligns probes to a diagonal line which runs through all of the dimensions. Again, a probe is placed at each limit, and the remaining $density - 2$ probes are evenly distributed between the boundary probes on this diagonal. The amount of probes required for this distribution is equal to the *density* parameter.

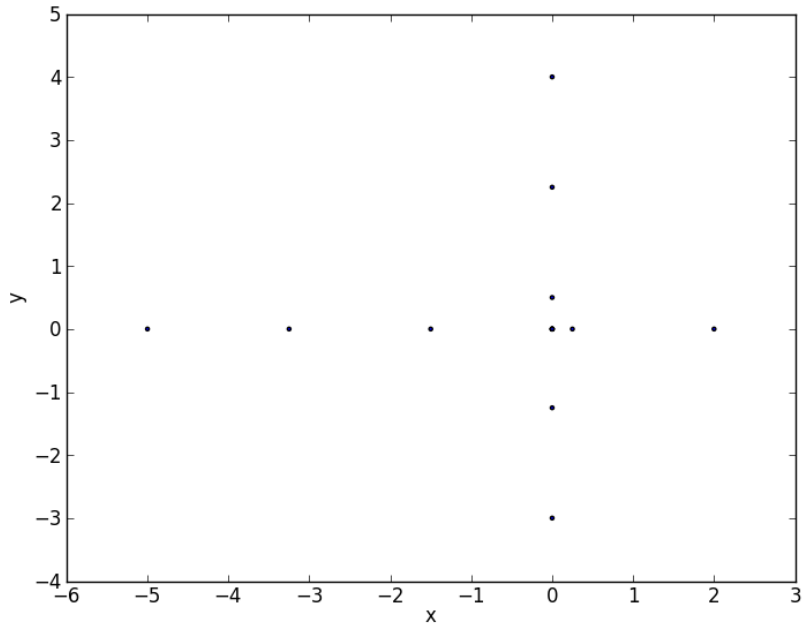


Figure 2-5: Uniform-On-Axis Probe Distribution

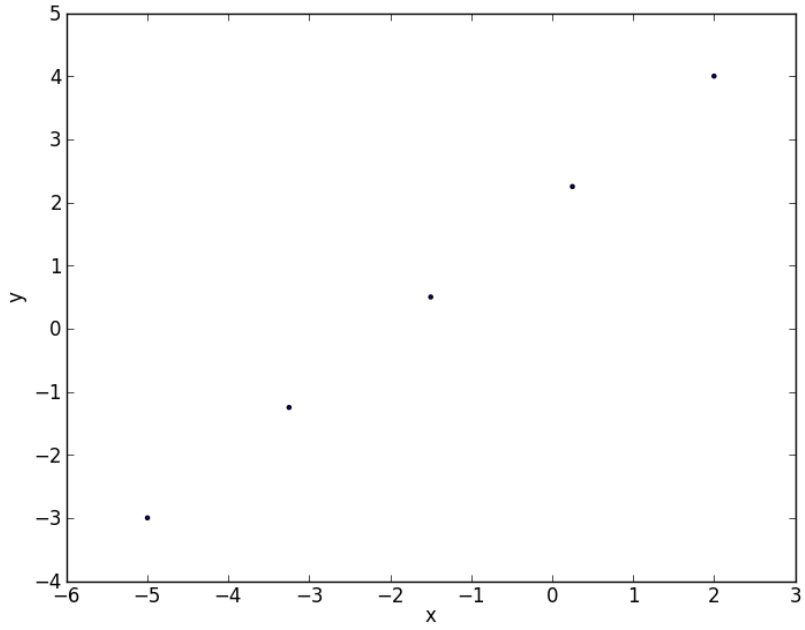


Figure 2-6: Uniform-On-Diagonal Probe Distribution

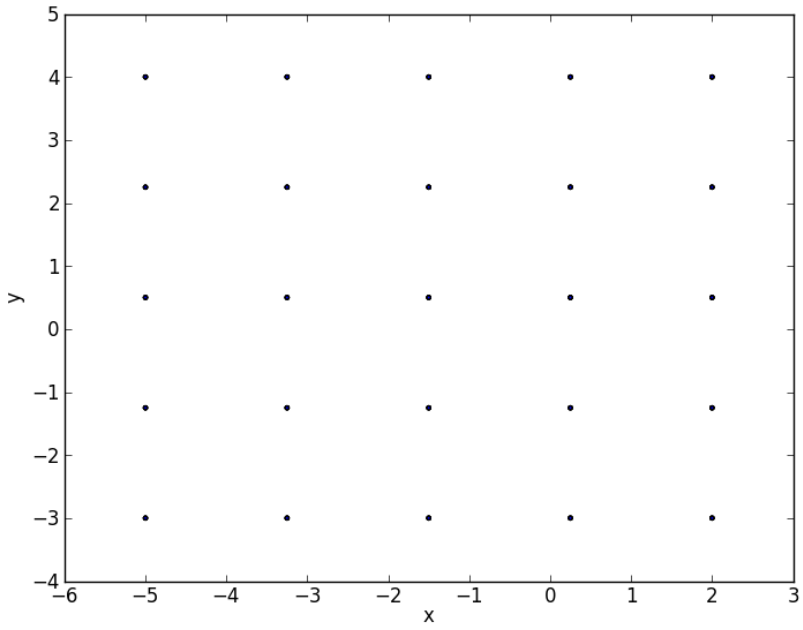


Figure 2-7: 2D Hypercube Probe Distribution

The *hypercube* (Fig. 2-7 - Fig. 2-10) distribution creates an evenly-spaced, n-dimensional grid, then aligns probes along each line intersection. This configuration requires the most probes and, as such, covers the search space much more than either of the other distribution patterns. The downside to this is that it requires a greater population of probes than other distributions and, therefore, a large number calculations. This distribution may be useful for running partial calculations. That is, running the algorithm for smaller periods of time (say 1,000 time intervals instead of 100,000), resetting the limits based on where the probes have gathered. This process would be repeated until the desired precision has been achieved. The amount of probes required for this distribution can be determined by raising the *density* parameter to the power of the *dim* parameter.

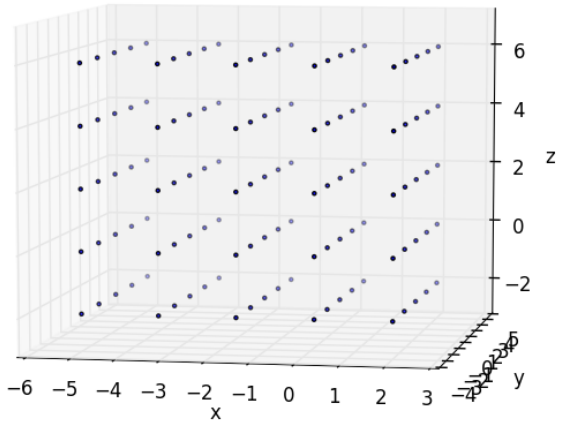


Figure 2-8: 3D Hypercube Probe Distribution (perspective a)

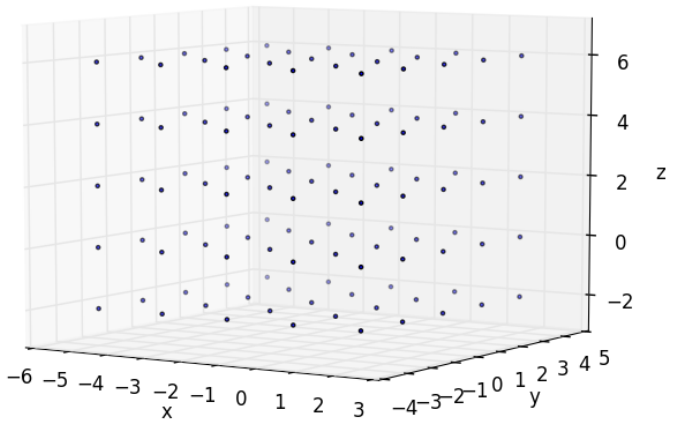


Figure 2-9: 3D Hypercube Probe Distribution (perspective b)

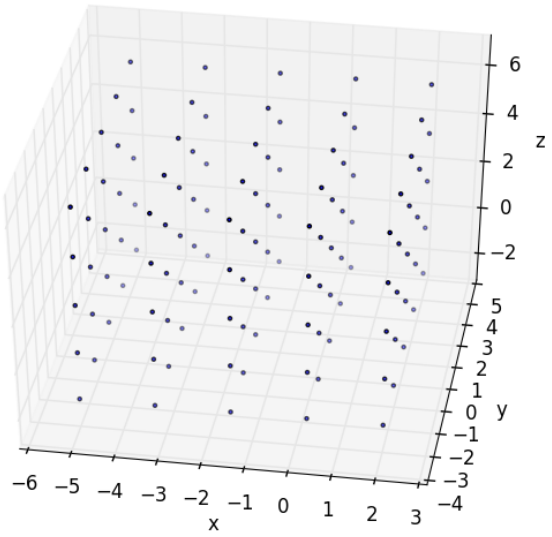


Figure 2-10: 3D Hypercube Probe Distribution (perspective c)

## Determining A Probe's Mass

The mass property of a probe is a scalar value which represents how well the probe's representative solution ($P_{\vec{p}_t}$) fits the objective function ($f$). The output of the objective function is directly mapped to the mass property of the probe. For this paper, all objective functions have been setup to be maximized, therefore, the largest output ($f(P_{\vec{p}})$) corresponds to the largest mass in the system. The greatest mass of the system represents that the most optimal solution can be found using the coordinates of the associated probe. If an objective function is setup to be maximized, but requires minimization, it may be modified to return opposite of its output ($-f(P_{\vec{p}})$). Thus, this implementation is useful for minimization problems as well.

## Updating A Probe's Position

As previously mentioned, a probe's dimensional coordinates coincide with the solution it represents; when a probe's position is updated, a new solution is explored. A probe's position is updated using its previous position and acceleration (2.1).

$$P_{\vec{p}t} = P_{\vec{p}_{t-1}} + \frac{1}{2} P_{\vec{a}_{t-1}} \tag{2.1}$$

It should be noted that this equation is a modified form of Newton's equation for the movement of physical objects (with constant acceleration) through space (2.2). It has been modified such that $\delta t = 1$ and the initial velocity has been removed.

$$\vec{r}_t = \vec{r}_0 + \vec{v}_0 \delta t + \frac{1}{2} \vec{a} \delta t^2 \tag{2.2}$$

There is a possibility that probes can relocate to a position which is no longer within the search space. In this case, the probe must be re-positioned within the search space [42] [43]. If one or more of the probe's coordinates is outside of the limit for that dimension, it will be placed at a distance located halfway between its previous position and the minimum or maximum (depending which limit has been violated) limit for that dimension. For probe coordinates which are greater than the upper limit (for that single dimension), the following equation (2.3) will be used to reassign the probe's positional coordinate for that dimension (other coordinates are left unchanged, provided they are within the limits for their individual dimensions).

$$P_{p_{t_d}} = P_{p_{t-1_d}} + \frac{max_d - P_{p_{t-1_d}}}{2} \tag{2.3}$$

Likewise, there is an equation which determines the re-positioning of the coordinate which violates a minimum limit for each dimension (2.4).

$$P_{p_{t_d}} = P_{p_{t-1_d}} - \frac{P_{p_{t-1_d}} - min_d}{2} \tag{2.4}$$

## Checking For Convergence

There are two possible scenarios in which the algorithm will stop running. The first scenario is reached if a predetermined amount of probe generations (the maximum time step) has been reached. The second scenario is achieved if the probes have converged upon a solution. The probes can be considered to have converged when at least one fifth (20%) of the probes are gathered within a neighborhood of some optimal probe. The neighborhood size is user defined in the *accuracy* parameter. The population of probes is evaluated for convergence each generation and precedes the *update acceleration* step. The evaluation has been placed at this point in the algorithm in order to avoid the lengthy operations involved with updating accelerations for a generation which will not make use of them (it has converged). If it has been determined that the probes have converged, then the probe with the greatest mass in this final population will be returned as the optimal solution.

## Updating A Probe's Acceleration

The acceleration of each probe is updated using a (heavily) modified version of the Newtonian equation for gravitational attraction (2.6) [44]. Since Newton's equation (2.5) describes objects in the physical world and CFO does not, many liberties can be taken to better achieve faster convergence in the computational world of CFO.

$$F = \frac{\gamma m_1 m_2}{r^2} \tag{2.5}$$

For a given time step ($t$) and dimension ($d$), a probe ($i$) has an acceleration of ($_iP_{a_{d_t}}$). This vector (2.6) will be used to determine probe $i$'s position during the

following time interval $(t+1)$ (2.1).

$$ {}_iP_{a_{d_t}} = \Sigma_{j=1}^{n} \frac{\gamma \cdot U({}_jP_{m_t} -_i P_{m_t}) \cdot ({}_jP_{m_t} -_i P_{m_t})^{\alpha} \cdot ({}_jP_{p_{d_t}} -_i P_{p_{d_t}})}{||({}_jP_{\vec{p}_t} -_i P_{\vec{p}_t})||^{\beta}} \tag{2.6} $$

where $i \neq j$ and $n$ is the total number of probes

The unit step function (2.7) ensures that a probe is only influenced by probes of greater or equal mass. This protects against the possibility that many probes of small mass pull a probe of greater mass off of an ideal solution. This also ensures that probes are not pushed away from other probes, due to negative differences in masses. Using the unit step function ensures that each probe is always moving to an area of greater probability for increasing the quality of its solution (its mass). If the unit step function returns a zero value, the acceleration calculation is broken for that probe of influence and the next probe's influence calculation is begun. This saves needless calculations from taking place if a zero multiplier exists in the numerator.

$$ U(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} \tag{2.7} $$

where $x =_j P_m -_i P_m$

for a pivot probe $i$ and some probe $j$

The second term in the numerator is the difference in mass between the influencing and pivot probe (the *delta mass*). The delta mass term determines the strength of the influence which probe $j$ induces on probe $i$. This term will always be a positive number, ensured by the unit step function.

The third term in the numerator works with the denominator to form a sort of unit vector which indicates the direction of the influence. The presence of exponents

on each term makes this different from typical unit vectors, which have an exponent value of 1.

*Alpha*, *beta*, and *gamma* are constants determined by the user before CFO is run. Certain sets of values for these constants may prove more beneficial than others. At the time of this writing, research is still required to determine which values are best, and for which applications, for *alpha*, *beta*, and *gamma*.

*Alpha* ($\alpha$) determines the effect which differences of mass will have when considering the influence of a probe. Lower values for $\alpha$ result in probes of nearly equal mass having little effect on each other. If the topology of the space being evaluated is largely uniform, one might wish to increase the value of $\alpha$ because it is unlikely that probes of greater mass will be able to enact strongly enough on the system to encourage convergence without requiring an extravagant runtime.

*Beta* ($\beta$), which is always 2 in Newton's equation, determines the deterioration of one probe's influence on the other as distance changes. Larger values for $\beta$ result in influence being less effective more quickly as probes separate. Lower values for $\beta$ allow probes to influence others from a greater distance.

*Gamma* ($\gamma$) serves the same purpose in the CFO equation as it does in Newton's, it represents the gravitational constant of the system. Higher values for $\gamma$ result in faster rates of acceleration, assuming no changes in $\beta$ and $\gamma$. Faster acceleration rates could be beneficial for exploring larger spaces, but could be detrimental to the convergence of the system or the accuracy of the solution.

The final alteration to the Newton's acceleration function (2.5) is an acceleration cap. This limits the distance that a probe is able to travel in order to adequately cover solutions within the search space. Without it, there is a possibility that a probe could

accelerate to such a level that it is continuously leaving the boundaries of the search space before exploring solutions on its trajectory. The acceleration of each probe is limited such that it may not be greater than 10% of the distance of the search space in each dimension (2.8).

$$P_{max\,a_d} = 0.1(upperLimit_d - lowerLimit_d) \tag{2.8}$$

# Chapter 3

# Python Objects in CFO

There is some debate as to how optimization algorithms might be implemented. Object-based implementations offer readability for computer scientists and programmers while array-based implementations might feel more familiar to those with a background in Mathematics. With this in mind, these two different implementations have been tested against each other to determine the impact, if any, that changes in the way that data is represented (whether in several disjoint arrays or encapsulated in an object) might have on the algorithm's runtime. Each implementation is an exact duplicate of the other, the only changes are in the initialization of object-representative structures and the specific calls necessary to reference the data in each structure.

As previously stated, the original structure (as described by Formato [19]) employs multiple disjoint arrays to organize and track the properties of the system of probes as they progressed through time. These *property arrays* contain information for each probe's acceleration, position, and mass at every time-interval.

Fig. 3-1 shows the structure of a single property array. Note that each property array is actually an array of arrays. The topmost array holds an array of probes for each time interval. For a given time interval, the array of probes is depicted

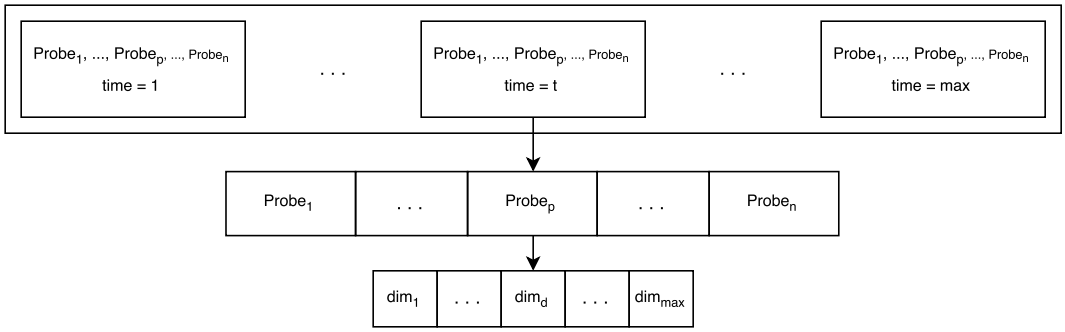


Figure 3-1: Property Array Structure

```
def createStructures(probeDist):
    .
    .
    .
    Accel     = [ [ [ 0 for d in range(dim) ] for p in range(numProbes) ] for t in range(max_time)]
    Mass      = [ [ 0 for p in range(numProbes) ] for t in range(max_time) ]
    Position  = [ [ [ 0 for d in range(dim) ] for p in range(numProbes) ] for t in range(max_time)]
    .
    .
    .
```

Figure 3-2: Initialization Of Property Arrays

in the middle array. For each probe element in this middle array, there exists a third (bottom-most) array which holds the actual values for that property in each dimension (ie. the acceleration factor for each dimension). The mass property array will contain only a single value in the bottom-most section of the figure because mass is a one-dimensional measure. Fig. 3-2 indicates the initialization of these arrays.

In the object-based implementation, rather than using disjoint arrays, a single $m$ x $n$ array is created. This array consists of $m$ elements (one for each probe) in each of the $n$ possible time intervals. Each probe's data is encapsulated in a computational object with properties for the probe's Acceleration, Mass, and Position. As in the array-based implementation, the Acceleration and Position data are still arrays with one floating-point value per dimension; the mass also remains a single floating point number.

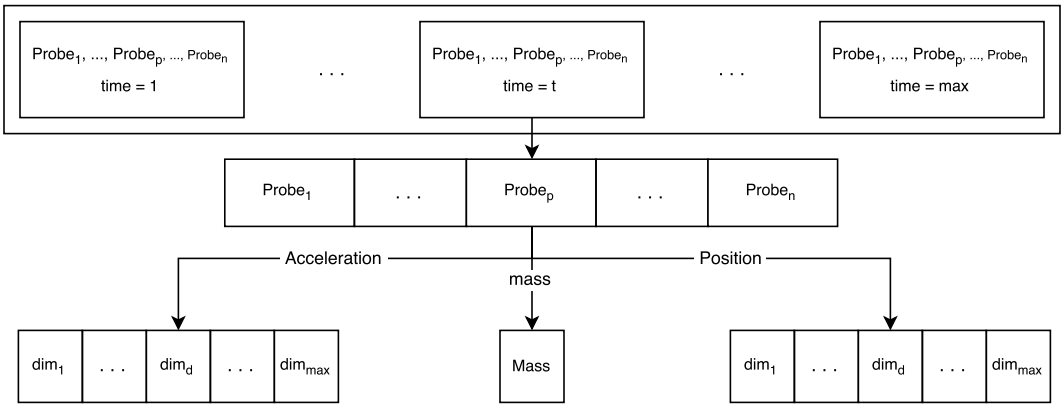


Figure 3-3: Object Implementation Structure

```
class probe(object):
    def __init__(self, _id):
        self.id          = _id
        self.position = [0.0] * dim
        self.accel       = [0.0] * dim
        self.mass        = 0.0E-7

def createStructures():
    .
    .
    .
    probes = [[probe(p) for p in range(numProbes)] for time in range(max_time)]
    .
    .
    .
```

Figure 3-4: Object Structure Initialization

Fig. 3-3 illustrates the entire structure for the object-based implementation of CFO. The same information can be accessed, but it is now arranged in within a single object for each probe. Modifying the attributes of a probe (such as implementing a multiplicity factor) only requires that the object definition includes a new property (*self.multiplicityFactor*) and code can be written to manipulate it as necessary. An array-based implementation requires another property array (Fig. 3-1) to be initialized before it can be used. Thus, in certain situations, encapsulating the data into an object may decrease the time necessary to implement the algorithm as well as increase readability.

## 3.1 Test Setup

Each implementation (array-based and object-based) is run on each of the (23) test functions. For each test function and each of the configurations in Table A.1 (for 2D functions) or Table A.2 (for nD functions) is tested. Each configuration is run 10 times, which results in 1940 test runs for each implementation. The *accuracy* parameter is set at 0.001 for each test. Since the accuracy of the solution is not of interest, the time limit is set to 1000 for 2D functions and 5000 for ND functions. This is relatively low in comparison to typical runs, which may require hundreds of thousands of time intervals to complete. Tests are implemented using the Python programming language and run on a computer with an Intel® Core™ i5-3210M CPU @ 2.50 GHz with 8 GB RAM.

## 3.2 Hypothesis

It is expected that the tests will reveal the object-based implementation to be the faster of the two. Encapsulating data into an object should allow the computer to reference the data faster due to the proximity of the final value array to the original reference. That is, that the computer will not have to drive into multiple arrays to retrieve data, but, once past the initial array which contains all probes for that time period (which is common to both implementations), the computer will simply pass the addresses of the object, then the address of the property before arriving at the final array which contains a single data value or value array (which is common to both implementations).

## 3.3 Runtimes

Table 3.1 provides a summary of the runtimes for each implementation. The average runtime is based on the 10 runs performed for each configuration of each test. $\Delta$ *runtime* indicates how much the average amount of time (in seconds) which is saved by running an array-based implementation vs an object-based implementation. One should note that, with the exception of a single configuration for test function 13, the array-based implementation achieves a runtime less than or equal to that of the object-based implementation. Instances where array-based data displays larger runtime than object-encapsulated data can be found when runtimes are less than 100 ms. These instances are inconsistent and, thus, have been attributed to system fluctuations and have not appeared in any test which runs longer than 100 ms.

**Function 01**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.96 ± 0.09 | 13.17 ± 0.10 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.76 ± 0.25 | 45.13 ± 0.36 | 5 | 10 | UOA | 1000 | -7.38 |
| 116.80 ± 0.45 | 135.38 ± 0.53 | 10 | 5 | UOA | 1000 | -18.58 |
| 18.42 ± 0.16 | 22.09 ± 0.13 | 3 | 50 | UOD | 1000 | -3.68 |
| 37.60 ± 0.17 | 44.87 ± 0.31 | 5 | 50 | UOD | 1000 | -7.27 |
| 112.74 ± 5.07 | 132.67 ± 4.86 | 10 | 50 | UOD | 1000 | -19.92 |
| 5.72 ± 0.05 | 6.79 ± 0.04 | 3 | 3 | Hyper | 1000 | -1.07 |
| 71.26 ± 0.62 | 85.79 ± 0.44 | 4 | 3 | Hyper | 1000 | -14.54 |

**Function 02**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.07 ± 0.14 | 64.48 ± 0.21 | 2 | 25 | UOA | 5000 | -11.40 |
| 53.08 ± 0.12 | 64.35 ± 0.13 | 2 | 50 | UOD | 5000 | -11.27 |
| 51.21 ± 0.15 | 62.13 ± 0.11 | 2 | 7 | Hyper | 5000 | -10.93 |

**Function 03**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.23 ± 0.12 | 64.48 ± 0.23 | 2 | 25 | UOA | 5000 | -11.25 |
| 53.61 ± 0.34 | 65.01 ± 0.30 | 2 | 50 | UOD | 5000 | -11.41 |
| 51.40 ± 0.16 | 62.19 ± 0.10 | 2 | 7 | Hyper | 5000 | -10.79 |

**Function 04**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 52.89 ± 0.12 | 63.96 ± 0.12 | 2 | 25 | UOA | 5000 | -11.06 |
| 52.86 ± 0.08 | 63.88 ± 0.05 | 2 | 50 | UOD | 5000 | -11.02 |
| 50.86 ± 0.06 | 61.42 ± 0.04 | 2 | 7 | Hyper | 5000 | -10.55 |

**Function 05**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.40 ± 0.07 | 64.41 ± 0.06 | 2 | 25 | UOA | 5000 | -11.01 |
| 53.41 ± 0.07 | 64.35 ± 0.09 | 2 | 50 | UOD | 5000 | -10.94 |
| 51.29 ± 0.05 | 61.82 ± 0.08 | 2 | 7 | Hyper | 5000 | -10.52 |

**Function 06**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 62.69 ± 0.15 | 74.23 ± 0.17 | 2 | 25 | UOA | 5000 | -11.54 |
| 57.60 ± 0.14 | 69.62 ± 0.12 | 2 | 50 | UOD | 5000 | -12.02 |
| 72.07 ± 0.28 | 83.65 ± 0.23 | 2 | 7 | Hyper | 5000 | -11.57 |

**Function 07**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.73 ± 0.15 | 64.98 ± 0.23 | 2 | 25 | UOA | 5000 | -11.25 |
| 54.25 ± 0.22 | 65.39 ± 0.27 | 2 | 50 | UOD | 5000 | -11.15 |
| 52.25 ± 0.13 | 62.88 ± 0.19 | 2 | 7 | Hyper | 5000 | -10.63 |

**Function 08**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.83 ± 0.12 | 13.09 ± 0.05 | 2 | 25 | UOA | 1000 | -2.26 |
| 37.36 ± 0.16 | 44.68 ± 0.11 | 5 | 10 | UOA | 1000 | -7.32 |
| 113.31 ± 0.21 | 131.62 ± 0.13 | 10 | 5 | UOA | 1000 | -18.31 |
| 18.17 ± 0.04 | 21.83 ± 0.04 | 3 | 50 | UOD | 1000 | -3.66 |
| 37.09 ± 0.08 | 44.28 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 109.38 ± 0.20 | 129.52 ± 0.34 | 10 | 50 | UOD | 1000 | -20.14 |
| 29.54 ± 0.07 | 35.58 ± 0.05 | 3 | 4 | Hyper | 1000 | -6.04 |
| 70.41 ± 0.08 | 84.69 ± 0.11 | 4 | 3 | Hyper | 1000 | -14.28 |

**Function 09**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.99 ± 0.12 | 65.21 ± 0.19 | 2 | 25 | UOA | 5000 | -11.22 |
| 53.95 ± 0.20 | 65.32 ± 0.24 | 2 | 50 | UOD | 5000 | -11.36 |
| 51.89 ± 0.17 | 62.86 ± 0.17 | 2 | 7 | Hyper | 5000 | -10.97 |

**Function 10**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 0.89 ± 0.04 | 0.94 ± 0.02 | 2 | 25 | UOA | 5000 | -0.04 |
| 0.73 ± 0.01 | 0.75 ± 0.01 | 2 | 50 | UOD | 5000 | -0.02 |
| 1.35 ± 0.01 | 1.50 ± 0.02 | 2 | 7 | Hyper | 5000 | -0.16 |

**Function 11**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.73 ± 0.04 | 12.97 ± 0.04 | 2 | 25 | UOA | 1000 | -2.24 |
| 37.31 ± 0.04 | 44.58 ± 0.06 | 5 | 10 | UOA | 1000 | -7.27 |
| 112.01 ± 0.14 | 132.65 ± 0.29 | 10 | 5 | UOA | 1000 | -20.64 |
| 18.17 ± 0.05 | 21.80 ± 0.04 | 3 | 50 | UOD | 1000 | -3.63 |
| 37.19 ± 0.05 | 44.40 ± 0.07 | 5 | 50 | UOD | 1000 | -7.21 |
| 110.02 ± 0.11 | 130.37 ± 0.08 | 10 | 50 | UOD | 1000 | -20.35 |
| 29.27 ± 0.05 | 35.33 ± 0.04 | 3 | 4 | Hyper | 1000 | -6.06 |
| 69.72 ± 0.08 | 83.91 ± 0.10 | 4 | 3 | Hyper | 1000 | -14.19 |

**Function 12**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.68 ± 0.05 | 12.92 ± 0.03 | 2 | 25 | UOA | 1000 | -2.24 |
| 36.83 ± 0.05 | 44.12 ± 0.07 | 5 | 10 | UOA | 1000 | -7.29 |
| 108.90 ± 0.13 | 130.57 ± 4.28 | 10 | 5 | UOA | 1000 | -21.66 |
| 17.98 ± 0.04 | 21.62 ± 0.05 | 3 | 50 | UOD | 1000 | -3.64 |
| 36.74 ± 0.08 | 43.93 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 108.25 ± 0.15 | 128.65 ± 0.21 | 10 | 50 | UOD | 1000 | -20.40 |
| 29.23 ± 0.07 | 35.33 ± 0.06 | 3 | 4 | Hyper | 1000 | -6.10 |
| 69.41 ± 0.09 | 83.63 ± 0.09 | 4 | 3 | Hyper | 1000 | -14.22 |

**Function 13**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 53.29 ± 0.20 | 65.19 ± 0.10 | 2 | 25 | UOA | 5000 | -11.90 |
| 0.54 ± 0.03 | 0.51 ± 0.03 | 2 | 50 | UOD | 5000 | 0.03 |
| 51.21 ± 0.10 | 62.27 ± 0.20 | 2 | 7 | Hyper | 5000 | -11.06 |

**Function 14**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.76 ± 0.03 | 12.97 ± 0.04 | 2 | 25 | UOA | 1000 | -2.20 |
| 37.02 ± 0.06 | 44.19 ± 0.09 | 5 | 10 | UOA | 1000 | -7.17 |
| 0.77 ± 0.02 | 0.82 ± 0.02 | 10 | 5 | UOA | 1000 | -0.05 |
| 18.15 ± 0.05 | 21.76 ± 0.06 | 3 | 50 | UOD | 1000 | -3.61 |
| 37.05 ± 0.06 | 44.21 ± 0.17 | 5 | 50 | UOD | 1000 | -7.16 |
| 109.59 ± 0.18 | 129.67 ± 0.12 | 10 | 50 | UOD | 1000 | -20.08 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 29.54 ± 0.05 | 35.49 ± 0.06 | 3 | 4 | Hyper | 1000 | -5.95 |
| 0.59 ± 0.01 | 0.65 ± 0.02 | 4 | 3 | Hyper | 1000 | -0.06 |

**Function 15**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.73 ± 0.04 | 12.93 ± 0.04 | 2 | 25 | UOA | 1000 | -2.19 |
| 0.31 ± 0.01 | 0.31 ± 0.01 | 5 | 10 | UOA | 1000 | 0.00 |
| 0.71 ± 0.02 | 0.75 ± 0.02 | 10 | 5 | UOA | 1000 | -0.03 |
| 18.30 ± 0.06 | 21.93 ± 0.06 | 3 | 50 | UOD | 1000 | -3.63 |
| 37.31 ± 0.04 | 44.50 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 110.12 ± 0.14 | 130.31 ± 0.17 | 10 | 50 | UOD | 1000 | -20.19 |
| 29.53 ± 0.10 | 35.45 ± 0.07 | 3 | 4 | Hyper | 1000 | -5.92 |
| 70.12 ± 0.06 | 84.21 ± 0.08 | 4 | 3 | Hyper | 1000 | -14.09 |

**Function 16**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.80 ± 0.06 | 12.99 ± 0.04 | 2 | 25 | UOA | 1000 | -2.19 |
| 37.08 ± 0.05 | 44.30 ± 0.08 | 5 | 10 | UOA | 1000 | -7.21 |
| 113.80 ± 0.20 | 131.79 ± 0.15 | 10 | 5 | UOA | 1000 | -17.99 |
| 18.13 ± 0.04 | 21.76 ± 0.06 | 3 | 50 | UOD | 1000 | -3.64 |
| 37.40 ± 1.07 | 44.76 ± 1.96 | 5 | 50 | UOD | 1000 | -7.36 |
| 108.94 ± 0.10 | 128.85 ± 0.12 | 10 | 50 | UOD | 1000 | -19.91 |
| 29.56 ± 0.07 | 35.59 ± 0.09 | 3 | 4 | Hyper | 1000 | -6.03 |
| 69.32 ± 0.04 | 83.28 ± 0.12 | 4 | 3 | Hyper | 1000 | -13.96 |

**Function 17**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.89 ± 0.04 | 13.09 ± 0.04 | 2 | 25 | UOA | 1000 | -2.20 |
| 37.28 ± 0.09 | 44.45 ± 0.07 | 5 | 10 | UOA | 1000 | -7.17 |
| 118.02 ± 0.07 | 136.87 ± 0.12 | 10 | 5 | UOA | 1000 | -18.85 |
| 18.18 ± 0.03 | 21.86 ± 0.07 | 3 | 50 | UOD | 1000 | -3.68 |
| 37.10 ± 0.03 | 44.21 ± 0.06 | 5 | 50 | UOD | 1000 | -7.11 |
| 109.60 ± 0.15 | 129.71 ± 0.13 | 10 | 50 | UOD | 1000 | -20.11 |
| 29.87 ± 0.07 | 35.87 ± 0.03 | 3 | 4 | Hyper | 1000 | -6.00 |
| 77.14 ± 0.09 | 92.53 ± 0.13 | 4 | 3 | Hyper | 1000 | -15.39 |

**Function 18**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.82 ± 0.05 | 13.02 ± 0.04 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.30 ± 0.07 | 44.46 ± 0.05 | 5 | 10 | UOA | 1000 | -7.16 |
| 114.95 ± 0.16 | 133.12 ± 0.14 | 10 | 5 | UOA | 1000 | -18.17 |
| 18.13 ± 0.05 | 21.74 ± 0.08 | 3 | 50 | UOD | 1000 | -3.61 |
| 37.02 ± 0.06 | 44.12 ± 0.05 | 5 | 50 | UOD | 1000 | -7.10 |
| 109.26 ± 0.08 | 129.10 ± 0.20 | 10 | 50 | UOD | 1000 | -19.84 |
| 29.69 ± 0.06 | 35.65 ± 0.04 | 3 | 4 | Hyper | 1000 | -5.97 |
| 71.08 ± 0.07 | 85.28 ± 0.11 | 4 | 3 | Hyper | 1000 | -14.20 |

**Function 19**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.87 ± 0.61 | 13.17 ± 0.62 | 2 | 25 | UOA | 1000 | -2.30 |
| 37.16 ± 0.11 | 44.49 ± 0.15 | 5 | 10 | UOA | 1000 | -7.33 |
| 111.36 ± 0.53 | 132.20 ± 0.23 | 10 | 5 | UOA | 1000 | -20.84 |
| 0.39 ± 0.02 | 0.44 ± 0.01 | 3 | 50 | UOD | 1000 | -0.05 |
| 0.75 ± 0.01 | 0.85 ± 0.01 | 5 | 50 | UOD | 1000 | -0.10 |
| 2.03 ± 0.02 | 2.33 ± 0.03 | 10 | 50 | UOD | 1000 | -0.30 |
| 1.54 ± 0.02 | 1.83 ± 0.02 | 3 | 4 | Hyper | 1000 | -0.29 |
| 2.40 ± 0.03 | 2.83 ± 0.03 | 4 | 3 | Hyper | 1000 | -0.44 |

**Function 20**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 6.60 ± 0.03 | 8.00 ± 0.04 | 2 | 25 | UOA | 1000 | -1.40 |
| 36.93 ± 0.22 | 44.35 ± 0.22 | 5 | 10 | UOA | 1000 | -7.42 |
| 111.42 ± 0.86 | 132.15 ± 0.32 | 10 | 5 | UOA | 1000 | -20.73 |
| 2.52 ± 0.01 | 3.00 ± 0.02 | 3 | 50 | UOD | 1000 | -0.48 |
| 8.12 ± 0.03 | 9.67 ± 0.04 | 5 | 50 | UOD | 1000 | -1.55 |
| 45.93 ± 0.18 | 54.67 ± 0.17 | 10 | 50 | UOD | 1000 | -8.74 |
| 18.91 ± 0.10 | 22.84 ± 0.10 | 3 | 4 | Hyper | 1000 | -3.93 |
| 69.55 ± 0.31 | 83.98 ± 0.36 | 4 | 3 | Hyper | 1000 | -14.43 |

**Function 21**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.77 ± 0.04 | 12.99 ± 0.05 | 2 | 25 | UOA | 1000 | -2.23 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 37.19 ± 0.21 | 44.58 ± 0.22 | 5 | 10 | UOA | 1000 | -7.39 |
| 114.11 ± 0.51 | 132.47 ± 0.32 | 10 | 5 | UOA | 1000 | -18.36 |
| 18.20 ± 0.08 | 21.86 ± 0.08 | 3 | 50 | UOD | 1000 | -3.66 |
| 37.40 ± 0.17 | 44.68 ± 0.18 | 5 | 50 | UOD | 1000 | -7.28 |
| 110.11 ± 0.28 | 131.06 ± 0.35 | 10 | 50 | UOD | 1000 | -20.95 |
| 29.74 ± 0.13 | 35.77 ± 0.15 | 3 | 4 | Hyper | 1000 | -6.03 |
| 71.01 ± 0.15 | 85.08 ± 0.30 | 4 | 3 | Hyper | 1000 | -14.07 |

**Function 22**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 10.76 ± 0.04 | 12.98 ± 0.08 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.42 ± 0.09 | 44.73 ± 0.14 | 5 | 10 | UOA | 1000 | -7.31 |
| 0.87 ± 0.02 | 0.92 ± 0.01 | 10 | 5 | UOA | 1000 | -0.05 |
| 18.19 ± 0.07 | 21.96 ± 0.10 | 3 | 50 | UOD | 1000 | -3.77 |
| 37.35 ± 0.15 | 44.82 ± 0.15 | 5 | 50 | UOD | 1000 | -7.47 |
| 110.18 ± 0.60 | 131.01 ± 0.41 | 10 | 50 | UOD | 1000 | -20.83 |
| 29.67 ± 0.14 | 35.84 ± 0.10 | 3 | 4 | Hyper | 1000 | -6.16 |
| 0.60 ± 0.01 | 0.67 ± 0.01 | 4 | 3 | Hyper | 1000 | -0.06 |

**Function 23**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Array | Object | dim | pD | IPD | step | runtime |
| 11.27 ± 0.07 | 13.54 ± 0.09 | 2 | 25 | UOA | 1000 | -2.27 |
| 39.98 ± 0.19 | 47.37 ± 0.22 | 5 | 10 | UOA | 1000 | -7.38 |
| 124.18 ± 0.63 | 142.93 ± 0.48 | 10 | 5 | UOA | 1000 | -18.75 |
| 19.23 ± 0.05 | 22.96 ± 0.09 | 3 | 50 | UOD | 1000 | -3.73 |
| 40.38 ± 0.69 | 47.89 ± 1.38 | 5 | 50 | UOD | 1000 | -7.51 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 122.41 ± 1.86 | 141.93 ± 0.40 | 10 | 50 | UOD | 1000 | -19.52 |
| 31.88 ± 0.10 | 38.24 ± 0.16 | 3 | 4 | Hyper | 1000 | -6.36 |
| 0.63 ± 0.01 | 0.69 ± 0.01 | 4 | 3 | Hyper | 1000 | -0.06 |

Table 3.1: Array Vs Object-Oriented Runtime Summary

## 3.4 Statistical Analysis of Results

After the runtimes are recorded, a Kolmogorov-Smirnov Test is run on each set of (10) runs for each configuration to determine the probability that runtimes adhere to a normal distribution. The results conclude that, for every configuration, the values fit a normal distribution with a probability of greater than 99%. Concluding that the results fit a normal distribution, a Student's T-Test is run to determine the probability that the average runtime of each implementation is equal to the other. When the t-statistic is low ($< 3$), the p-value represents the probability supporting the null hypothesis (that there is no difference in the averages); alternatively, when the t-statistic is high ($\geq 3$), the p-value represents the probability that the null hypothesis can be rejected. The results of this test indicate that the runtime of each implementation is not equal (Table C).

## 3.5 Conclusions

As shown in Table 3.1, a measurable difference exists between the runtime of array and object-based implementations of CFO. Contrary to the hypothesis, runtimes for array-based implementations are measurably faster than object-based implementations for the Python language. For these test cases, it can be observed that the

average runtime of object-based data typically achieves a 15-20% slower runtime when compared to the coinciding array-based data implementation. Of course, if readability is the priority, the trade off (in terms of runtime) may be extrapolated from these results.

Observing the results of the tests described above, further research was conducted to determine why an object references slower than an array. Unlike lower level languages, such as C++, Python objects do not require the programmer to write a header for the custom classes which they create. Instead, a generic structure serves as a template for all object classes. This structure involves properties which describe the module that contains the class definition, the name of the class, base classes which are being extended, and a dictionary which contains entries for each of the classes methods and object attributes. It is this final property which may be the reason why object-encapsulated data is referenced more slowly than data from built-in types. When an object is called, after it is located, its dictionary must be searched for the property of interest, the availability (access rights) of the property must be determined, then the data itself can be returned. It is these extra steps which are likely responsible for the difference in runtime between the two implementations tested above.

# Chapter 4

# Multiplicity Factor

Due to its deterministic nature, CFO only requires a single run to achieve its result. This does not imply that this run may not be computationally intensive. Unfortunately, it is often the case that a typical run will involve an enormous amount of operations, most of which occur in the *Update Probe Acceleration* portion of the algorithm. Since each probe must calculate each other probe's position to itself, this requires $n(n-1)$ calculations for this step alone. Implementing a *multiplicity factor* shows that the number of operations in the *update acceleration* (as well as *update probe positions* and *update probe masses* steps, but to a lesser degree) may be greatly reduced.

A *multiplicity factor* utilizes the idea of combining probes which are within a given neighborhood each other. If a number of probes are within a *combination neighborhood* of each other (determined by the *accuracy* parameter specified by the user), they are combined to a single probe. This single probe is assigned a multiplicity factor equal to the amount of probes within the combination neighborhood and takes on the mass of the most optimal probe in the neighborhood. In order to implement this feature, an additional *Factor* property array is created. This array has a similar structure to the Mass property array in that it maps down to a one dimensional value. The only difference is that the Factor property array maps to a Natural number value

for each probe, whereas the Mass property array maps to a floating point number.

| **Population** | **Probe Comparisons** | **Comparisons Saved** |
|---|---|---|
| $n$ | $n(n-1) = n^2 - n$ | - |
| $n-1$ | $(n-1)(n-2) = n^2 - 3n + 3$ | $2n - 2$ |
| $n-2$ | $(n-2)(n-3) = n^2 - 5n + 6$ | $4n - 6$ |
| $n-3$ | $(n-3)(n-4) = n^2 - 7n + 12$ | $6n - 12$ |
| $n-i$ | $(n-i)(n-i-1) = n^2 - 2ni - n + i^2 + i$ | $2in - i(i+1)$ |

Table 4.1: Operations Saved When Combining Probes

## 4.1 Modified Algorithm & Implementation

Since probes may be combined, subsequent generations may not have the same number of elements as the previous generation. Therefore, the traditional array-based implementation has been altered to avoid null references. This change involves a number of modifications to allow for the shrinking size of populations.

First, rather than create property arrays with elements for each possible time step, property arrays are created with elements for only two time steps. The arrays represent the equivalence class modulus 2 of the enumeration for each time step. That is, even-interval time steps are stored in the $0^{th}$ ($[0]_2$) element and odd-interval time steps are stored in the $1^{st}$ ($[1]_2$) element. This takes advantage of the mathematical principle of equivalence classes and modular arithmetic. This modified algorithm can be see in Fig. 4.1. This modification also saves a great deal of memory, since the three property arrays now consist of $2n$ time step elements rather than the $tn$ time step elements (where $n$ is the number of probes and $t$ is the maximum amount of time steps allowed).

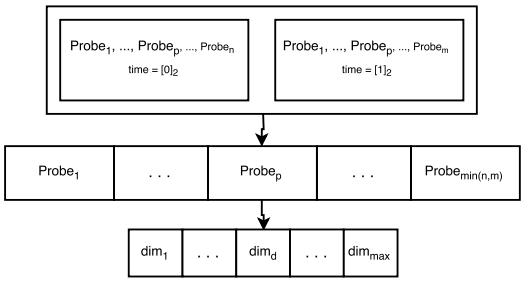


Figure 4-1: Modified Property Array Structure

Second, now that only two time step elements exist for each property array, the change in population size must be accounted for. This is achieved by initializing the next time step element of each property array to an empty array, then filling the probes in after the probe combination step.

```
1: determine amount of probes
2: create structures
3: initialize probe positions
4: compute initial masses
5: compute initial accelerations
6: for time = 1 until time = maximum do
7:     combine probes
8:     update probe positions
9:     update probe masses
10:    if probes have converged then
11:        break
12:    end if
13:    update probe accelerations
14: end for
```

Algorithm 2: Pseudocode For CFO Algorithm With Multiplicity Factor

The probe positions are evaluated and, if necessary, combined in the *combine probes* step of the algorithm. This is performed by a single nested for-loop. The outermost loop iterates through each of the probes probes of the current generation. Let this outer loop element be element $i$. The $i^{th}$ element serves as a pivot and is evaluated

against each of the $n - i$ elements in the generation to determine whether the probes are located within a *combination neighborhood* of each other. If a probe has already undergone the combination process and has been combined with another probe, then its multiplicity factor will have a zero value. In this case, it will absent from further evaluations to prevent it from being combined more than once.

For instance, if a run has a specified accuracy of 0.001 and $m$ probes are located within a distance of 0.001 units of each other, then these probes are combined into a single probe. The resulting probe will be located at the position of the best fit probe within the combination group. The combined group of probes now operates as a single probe with a multiplicity factor of $m$. Subsequent generations now operate on a population of $n - m + 1$ probes and, thus, the *Update Probe Accelerations* step will require $(n - m + 1)(n - m)$ operations. Table 4 illustrates the amount of operations saved in the *Update Probe Accelerations* step as probes are combined.

After the combine probes step has occurred, a modified version of the update probe positions step is performed. This step is modified to create new property arrays for the next generation. Each property array's size is based on the Factor property array. If a probe exhibits a zero value in the Factor property array, this indicates that it has been combined with another probe and will not be brought forward into the next generation.

## 4.2 Test Setup

As with the Array vs. Object-Oriented tests, each implementation is run each of of the (23) test functions. For each test function, each of the configurations in Table A.1 (for 2D functions) or Table A.2 (for nD functions) is tested. Each configuration is run 10 times, this results in 1940 test runs for each implementation. The *accuracy*

parameter is set to 0.001 for each test. Since the accuracy of the solution is not of interest, the time limit is set at 1000 for 2D functions and 5000 for ND functions. This is relatively low in comparison to typical runs, which may require hundreds of thousands of time intervals to complete. Tests are implemented using the Python programming language and run on a computer with an Intel® Core™ i5-3210M CPU @ 2.50 GHz with 8 GB RAM.

## 4.3 Runtimes

**Function 01**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.96 ± 0.09 | 13.17 ± 0.10 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.76 ± 0.25 | 45.13 ± 0.36 | 5 | 10 | UOA | 1000 | -7.38 |
| 116.80 ± 0.45 | 135.38 ± 0.53 | 10 | 5 | UOA | 1000 | -18.58 |
| 18.42 ± 0.16 | 22.09 ± 0.13 | 3 | 50 | UOD | 1000 | -3.68 |
| 37.60 ± 0.17 | 44.87 ± 0.31 | 5 | 50 | UOD | 1000 | -7.27 |
| 112.74 ± 5.07 | 132.67 ± 4.86 | 10 | 50 | UOD | 1000 | -19.92 |
| 5.72 ± 0.05 | 6.79 ± 0.04 | 3 | 3 | Hyper | 1000 | -1.07 |
| 71.26 ± 0.62 | 85.79 ± 0.44 | 4 | 3 | Hyper | 1000 | -14.54 |

**Function 02**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.07 ± 0.14 | 64.48 ± 0.21 | 2 | 25 | UOA | 5000 | -11.40 |
| 53.08 ± 0.12 | 64.35 ± 0.13 | 2 | 50 | UOD | 5000 | -11.27 |
| 51.21 ± 0.15 | 62.13 ± 0.11 | 2 | 7 | Hyper | 5000 | -10.93 |

**Function 03**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.23 ± 0.12 | 64.48 ± 0.23 | 2 | 25 | UOA | 5000 | -11.25 |
| 53.61 ± 0.34 | 65.01 ± 0.30 | 2 | 50 | UOD | 5000 | -11.41 |
| 51.40 ± 0.16 | 62.19 ± 0.10 | 2 | 7 | Hyper | 5000 | -10.79 |

**Function 04**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 52.89 ± 0.12 | 63.96 ± 0.12 | 2 | 25 | UOA | 5000 | -11.06 |
| 52.86 ± 0.08 | 63.88 ± 0.05 | 2 | 50 | UOD | 5000 | -11.02 |
| 50.86 ± 0.06 | 61.42 ± 0.04 | 2 | 7 | Hyper | 5000 | -10.55 |

**Function 05**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.40 ± 0.07 | 64.41 ± 0.06 | 2 | 25 | UOA | 5000 | -11.01 |
| 53.41 ± 0.07 | 64.35 ± 0.09 | 2 | 50 | UOD | 5000 | -10.94 |
| 51.29 ± 0.05 | 61.82 ± 0.08 | 2 | 7 | Hyper | 5000 | -10.52 |

**Function 06**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 62.69 ± 0.15 | 74.23 ± 0.17 | 2 | 25 | UOA | 5000 | -11.54 |
| 57.60 ± 0.14 | 69.62 ± 0.12 | 2 | 50 | UOD | 5000 | -12.02 |
| 72.07 ± 0.28 | 83.65 ± 0.23 | 2 | 7 | Hyper | 5000 | -11.57 |

**Function 07**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.73 ± 0.15 | 64.98 ± 0.23 | 2 | 25 | UOA | 5000 | -11.25 |
| 54.25 ± 0.22 | 65.39 ± 0.27 | 2 | 50 | UOD | 5000 | -11.15 |
| 52.25 ± 0.13 | 62.88 ± 0.19 | 2 | 7 | Hyper | 5000 | -10.63 |

**Function 08**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.83 ± 0.12 | 13.09 ± 0.05 | 2 | 25 | UOA | 1000 | -2.26 |
| 37.36 ± 0.16 | 44.68 ± 0.11 | 5 | 10 | UOA | 1000 | -7.32 |
| 113.31 ± 0.21 | 131.62 ± 0.13 | 10 | 5 | UOA | 1000 | -18.31 |
| 18.17 ± 0.04 | 21.83 ± 0.04 | 3 | 50 | UOD | 1000 | -3.66 |
| 37.09 ± 0.08 | 44.28 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 109.38 ± 0.20 | 129.52 ± 0.34 | 10 | 50 | UOD | 1000 | -20.14 |
| 29.54 ± 0.07 | 35.58 ± 0.05 | 3 | 4 | Hyper | 1000 | -6.04 |
| 70.41 ± 0.08 | 84.69 ± 0.11 | 4 | 3 | Hyper | 1000 | -14.28 |

**Function 09**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.99 ± 0.12 | 65.21 ± 0.19 | 2 | 25 | UOA | 5000 | -11.22 |
| 53.95 ± 0.20 | 65.32 ± 0.24 | 2 | 50 | UOD | 5000 | -11.36 |
| 51.89 ± 0.17 | 62.86 ± 0.17 | 2 | 7 | Hyper | 5000 | -10.97 |

**Function 10**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 0.89 ± 0.04 | 0.94 ± 0.02 | 2 | 25 | UOA | 5000 | -0.04 |
| 0.73 ± 0.01 | 0.75 ± 0.01 | 2 | 50 | UOD | 5000 | -0.02 |
| 1.35 ± 0.01 | 1.50 ± 0.02 | 2 | 7 | Hyper | 5000 | -0.16 |

**Function 11**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.73 ± 0.04 | 12.97 ± 0.04 | 2 | 25 | UOA | 1000 | -2.24 |
| 37.31 ± 0.04 | 44.58 ± 0.06 | 5 | 10 | UOA | 1000 | -7.27 |
| 112.01 ± 0.14 | 132.65 ± 0.29 | 10 | 5 | UOA | 1000 | -20.64 |
| 18.17 ± 0.05 | 21.80 ± 0.04 | 3 | 50 | UOD | 1000 | -3.63 |
| 37.19 ± 0.05 | 44.40 ± 0.07 | 5 | 50 | UOD | 1000 | -7.21 |
| 110.02 ± 0.11 | 130.37 ± 0.08 | 10 | 50 | UOD | 1000 | -20.35 |
| 29.27 ± 0.05 | 35.33 ± 0.04 | 3 | 4 | Hyper | 1000 | -6.06 |
| 69.72 ± 0.08 | 83.91 ± 0.10 | 4 | 3 | Hyper | 1000 | -14.19 |

**Function 12**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.68 ± 0.05 | 12.92 ± 0.03 | 2 | 25 | UOA | 1000 | -2.24 |
| 36.83 ± 0.05 | 44.12 ± 0.07 | 5 | 10 | UOA | 1000 | -7.29 |
| 108.90 ± 0.13 | 130.57 ± 4.28 | 10 | 5 | UOA | 1000 | -21.66 |
| 17.98 ± 0.04 | 21.62 ± 0.05 | 3 | 50 | UOD | 1000 | -3.64 |
| 36.74 ± 0.08 | 43.93 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 108.25 ± 0.15 | 128.65 ± 0.21 | 10 | 50 | UOD | 1000 | -20.40 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 29.23 ± 0.07 | 35.33 ± 0.06 | 3 | 4 | Hyper | 1000 | -6.10 |
| 69.41 ± 0.09 | 83.63 ± 0.09 | 4 | 3 | Hyper | 1000 | -14.22 |

**Function 13**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 53.29 ± 0.20 | 65.19 ± 0.10 | 2 | 25 | UOA | 5000 | -11.90 |
| 0.54 ± 0.03 | 0.51 ± 0.03 | 2 | 50 | UOD | 5000 | 0.03 |
| 51.21 ± 0.10 | 62.27 ± 0.20 | 2 | 7 | Hyper | 5000 | -11.06 |

**Function 14**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.76 ± 0.03 | 12.97 ± 0.04 | 2 | 25 | UOA | 1000 | -2.20 |
| 37.02 ± 0.06 | 44.19 ± 0.09 | 5 | 10 | UOA | 1000 | -7.17 |
| 0.77 ± 0.02 | 0.82 ± 0.02 | 10 | 5 | UOA | 1000 | -0.05 |
| 18.15 ± 0.05 | 21.76 ± 0.06 | 3 | 50 | UOD | 1000 | -3.61 |
| 37.05 ± 0.06 | 44.21 ± 0.17 | 5 | 50 | UOD | 1000 | -7.16 |
| 109.59 ± 0.18 | 129.67 ± 0.12 | 10 | 50 | UOD | 1000 | -20.08 |
| 29.54 ± 0.05 | 35.49 ± 0.06 | 3 | 4 | Hyper | 1000 | -5.95 |
| 0.59 ± 0.01 | 0.65 ± 0.02 | 4 | 3 | Hyper | 1000 | -0.06 |

**Function 15**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.73 ± 0.04 | 12.93 ± 0.04 | 2 | 25 | UOA | 1000 | -2.19 |
| 0.31 ± 0.01 | 0.31 ± 0.01 | 5 | 10 | UOA | 1000 | 0.00 |
| 0.71 ± 0.02 | 0.75 ± 0.02 | 10 | 5 | UOA | 1000 | -0.03 |
| 18.30 ± 0.06 | 21.93 ± 0.06 | 3 | 50 | UOD | 1000 | -3.63 |

| | | | | | | |
|---|---|---|---|---|---|---|
| 37.31 ± 0.04 | 44.50 ± 0.04 | 5 | 50 | UOD | 1000 | -7.19 |
| 110.12 ± 0.14 | 130.31 ± 0.17 | 10 | 50 | UOD | 1000 | -20.19 |
| 29.53 ± 0.10 | 35.45 ± 0.07 | 3 | 4 | Hyper | 1000 | -5.92 |
| 70.12 ± 0.06 | 84.21 ± 0.08 | 4 | 3 | Hyper | 1000 | -14.09 |

**Function 16**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.80 ± 0.06 | 12.99 ± 0.04 | 2 | 25 | UOA | 1000 | -2.19 |
| 37.08 ± 0.05 | 44.30 ± 0.08 | 5 | 10 | UOA | 1000 | -7.21 |
| 113.80 ± 0.20 | 131.79 ± 0.15 | 10 | 5 | UOA | 1000 | -17.99 |
| 18.13 ± 0.04 | 21.76 ± 0.06 | 3 | 50 | UOD | 1000 | -3.64 |
| 37.40 ± 1.07 | 44.76 ± 1.96 | 5 | 50 | UOD | 1000 | -7.36 |
| 108.94 ± 0.10 | 128.85 ± 0.12 | 10 | 50 | UOD | 1000 | -19.91 |
| 29.56 ± 0.07 | 35.59 ± 0.09 | 3 | 4 | Hyper | 1000 | -6.03 |
| 69.32 ± 0.04 | 83.28 ± 0.12 | 4 | 3 | Hyper | 1000 | -13.96 |

**Function 17**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.89 ± 0.04 | 13.09 ± 0.04 | 2 | 25 | UOA | 1000 | -2.20 |
| 37.28 ± 0.09 | 44.45 ± 0.07 | 5 | 10 | UOA | 1000 | -7.17 |
| 118.02 ± 0.07 | 136.87 ± 0.12 | 10 | 5 | UOA | 1000 | -18.85 |
| 18.18 ± 0.03 | 21.86 ± 0.07 | 3 | 50 | UOD | 1000 | -3.68 |
| 37.10 ± 0.03 | 44.21 ± 0.06 | 5 | 50 | UOD | 1000 | -7.11 |
| 109.60 ± 0.15 | 129.71 ± 0.13 | 10 | 50 | UOD | 1000 | -20.11 |
| 29.87 ± 0.07 | 35.87 ± 0.03 | 3 | 4 | Hyper | 1000 | -6.00 |
| 77.14 ± 0.09 | 92.53 ± 0.13 | 4 | 3 | Hyper | 1000 | -15.39 |

**Function 18**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.82 ± 0.05 | 13.02 ± 0.04 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.30 ± 0.07 | 44.46 ± 0.05 | 5 | 10 | UOA | 1000 | -7.16 |
| 114.95 ± 0.16 | 133.12 ± 0.14 | 10 | 5 | UOA | 1000 | -18.17 |
| 18.13 ± 0.05 | 21.74 ± 0.08 | 3 | 50 | UOD | 1000 | -3.61 |
| 37.02 ± 0.06 | 44.12 ± 0.05 | 5 | 50 | UOD | 1000 | -7.10 |
| 109.26 ± 0.08 | 129.10 ± 0.20 | 10 | 50 | UOD | 1000 | -19.84 |
| 29.69 ± 0.06 | 35.65 ± 0.04 | 3 | 4 | Hyper | 1000 | -5.97 |
| 71.08 ± 0.07 | 85.28 ± 0.11 | 4 | 3 | Hyper | 1000 | -14.20 |

**Function 19**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.87 ± 0.61 | 13.17 ± 0.62 | 2 | 25 | UOA | 1000 | -2.30 |
| 37.16 ± 0.11 | 44.49 ± 0.15 | 5 | 10 | UOA | 1000 | -7.33 |
| 111.36 ± 0.53 | 132.20 ± 0.23 | 10 | 5 | UOA | 1000 | -20.84 |
| 0.39 ± 0.02 | 0.44 ± 0.01 | 3 | 50 | UOD | 1000 | -0.05 |
| 0.75 ± 0.01 | 0.85 ± 0.01 | 5 | 50 | UOD | 1000 | -0.10 |
| 2.03 ± 0.02 | 2.33 ± 0.03 | 10 | 50 | UOD | 1000 | -0.30 |
| 1.54 ± 0.02 | 1.83 ± 0.02 | 3 | 4 | Hyper | 1000 | -0.29 |
| 2.40 ± 0.03 | 2.83 ± 0.03 | 4 | 3 | Hyper | 1000 | -0.44 |

**Function 20**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 6.60 ± 0.03 | 8.00 ± 0.04 | 2 | 25 | UOA | 1000 | -1.40 |
| 36.93 ± 0.22 | 44.35 ± 0.22 | 5 | 10 | UOA | 1000 | -7.42 |
| 111.42 ± 0.86 | 132.15 ± 0.32 | 10 | 5 | UOA | 1000 | -20.73 |
| 2.52 ± 0.01 | 3.00 ± 0.02 | 3 | 50 | UOD | 1000 | -0.48 |
| 8.12 ± 0.03 | 9.67 ± 0.04 | 5 | 50 | UOD | 1000 | -1.55 |
| 45.93 ± 0.18 | 54.67 ± 0.17 | 10 | 50 | UOD | 1000 | -8.74 |
| 18.91 ± 0.10 | 22.84 ± 0.10 | 3 | 4 | Hyper | 1000 | -3.93 |
| 69.55 ± 0.31 | 83.98 ± 0.36 | 4 | 3 | Hyper | 1000 | -14.43 |

**Function 21**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.77 ± 0.04 | 12.99 ± 0.05 | 2 | 25 | UOA | 1000 | -2.23 |
| 37.19 ± 0.21 | 44.58 ± 0.22 | 5 | 10 | UOA | 1000 | -7.39 |
| 114.11 ± 0.51 | 132.47 ± 0.32 | 10 | 5 | UOA | 1000 | -18.36 |
| 18.20 ± 0.08 | 21.86 ± 0.08 | 3 | 50 | UOD | 1000 | -3.66 |
| 37.40 ± 0.17 | 44.68 ± 0.18 | 5 | 50 | UOD | 1000 | -7.28 |
| 110.11 ± 0.28 | 131.06 ± 0.35 | 10 | 50 | UOD | 1000 | -20.95 |
| 29.74 ± 0.13 | 35.77 ± 0.15 | 3 | 4 | Hyper | 1000 | -6.03 |
| 71.01 ± 0.15 | 85.08 ± 0.30 | 4 | 3 | Hyper | 1000 | -14.07 |

**Function 22**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 10.76 ± 0.04 | 12.98 ± 0.08 | 2 | 25 | UOA | 1000 | -2.21 |
| 37.42 ± 0.09 | 44.73 ± 0.14 | 5 | 10 | UOA | 1000 | -7.31 |
| 0.87 ± 0.02 | 0.92 ± 0.01 | 10 | 5 | UOA | 1000 | -0.05 |
| 18.19 ± 0.07 | 21.96 ± 0.10 | 3 | 50 | UOD | 1000 | -3.77 |
| 37.35 ± 0.15 | 44.82 ± 0.15 | 5 | 50 | UOD | 1000 | -7.47 |
| 110.18 ± 0.60 | 131.01 ± 0.41 | 10 | 50 | UOD | 1000 | -20.83 |
| 29.67 ± 0.14 | 35.84 ± 0.10 | 3 | 4 | Hyper | 1000 | -6.16 |
| 0.60 ± 0.01 | 0.67 ± 0.01 | 4 | 3 | Hyper | 1000 | -0.06 |

**Function 23**

| Avg Runtime (seconds) $\pm\sigma$ | | | | | max | $\Delta$ |
|---|---|---|---|---|---|---|
| Standard | Multiplicity | dim | pD | IPD | step | runtime |
| 11.27 ± 0.07 | 13.54 ± 0.09 | 2 | 25 | UOA | 1000 | -2.27 |
| 39.98 ± 0.19 | 47.37 ± 0.22 | 5 | 10 | UOA | 1000 | -7.38 |
| 124.18 ± 0.63 | 142.93 ± 0.48 | 10 | 5 | UOA | 1000 | -18.75 |
| 19.23 ± 0.05 | 22.96 ± 0.09 | 3 | 50 | UOD | 1000 | -3.73 |
| 40.38 ± 0.69 | 47.89 ± 1.38 | 5 | 50 | UOD | 1000 | -7.51 |
| 122.41 ± 1.86 | 141.93 ± 0.40 | 10 | 50 | UOD | 1000 | -19.52 |
| 31.88 ± 0.10 | 38.24 ± 0.16 | 3 | 4 | Hyper | 1000 | -6.36 |
| 0.63 ± 0.01 | 0.69 ± 0.01 | 4 | 3 | Hyper | 1000 | -0.06 |

Table 4.2: Array Vs Object-Oriented Runtime Summary

## 4.4 Statistical Analysis of Results

As with the tests in the previous chapter, after runtimes are recorded, a Kolmogorov-Smirnov Test is run on each set of (10) runs for each configuration to determine the probability that runtimes adhere to a normal distribution. The results conclude that, for every configuration, the values fit a normal distribution with a probability of greater than 99%. Concluding that the results fit a normal distribution, a Student's T-Test is run to determine the probability that the average runtime of each implementation is equal to the other. When the t-statistic is low ($< 3$), the p-value represents the probability supporting the null hypothesis (that there is no difference in the averages); alternatively, when the t-statistic is high ($\geq 3$), the p-value represents the probability that the null hypothesis can be rejected. The results of this test indicate that the runtime of each implementation is not equal (Table D).

While combining probes carries the possibility of saving a great deal of calculations, it requires some additional calculations of its own to determine determine whether probes are within the *combination neighborhood*. These calculations add operations which add to the algorithm's runtime. If probes are not combined, the resulting runtime will be longer than without this step, but, as observed in Table 4.2 (Test Function 13), it can be seen that these calculations are negligible, especially when weighed against the possible savings. The configurations in Table 4.2 which did not converge can be found by noticing that the final step is equal to the maximum step parameter.

## 4.5 Probe Convergence And Initial Population Size

Additional tests are run to indicate the amount of combinations which take place during a given run. Using test function 01 and a maximum runtime of 500,000 steps,

Table 4.5 displays the initial amount of probes, the final amount of probes, and the step in which the algorithm converged. Note that an initial population of 1024 probes ran in less steps than an initial population of 100 probes. This may be attributed to the initial proximity of one probe to another. The more saturated a search space becomes, the amount of probe combinations should be expected to increase.

| # Probes | | Convergence |
|---|---|---|
| $t_0$ | $t_n$ | step ($t_n$) |
| 25 | 19 | 120 |
| 49 | 24 | 272 |
| 100 | 18 | 640 |
| 1024 | 29 | 594 |

Table 4.3: Multiplicity Factor Combination Summary For Test Function 01

Fig. 4-2 shows a hypercube arrangement of 1024 probes (each with a multiplicity factor of 1) at the first time step. Fig. 4-3 shows these same probes after 575 time steps. While each of the 1024 probes is still accounted for via the multiplicity factor, the number of calculations is far lower than when each probe was individually present. This has the possibility of an exponential decrease in runtime for each step as probes converge (Fig. 4-4).

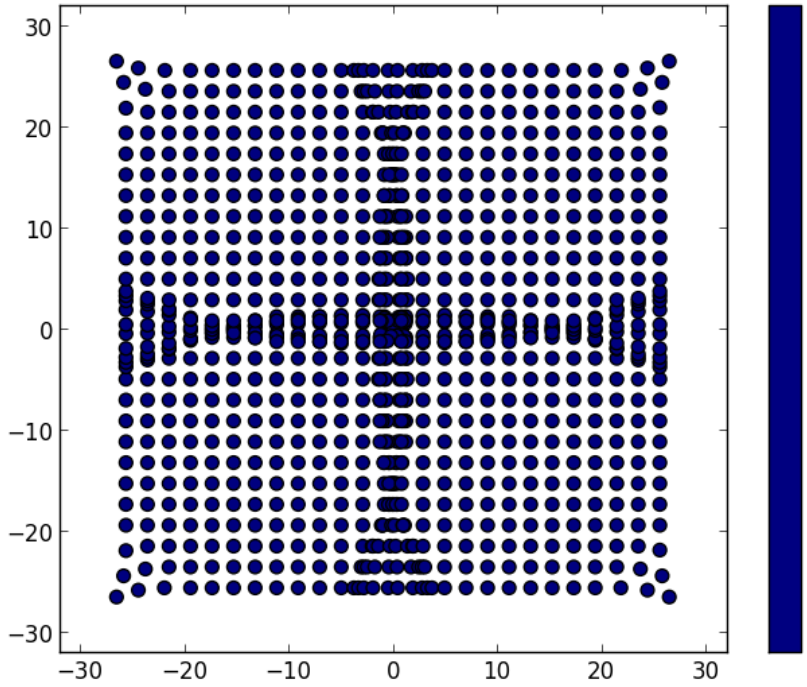


Figure 4-2: Probes At Step 1

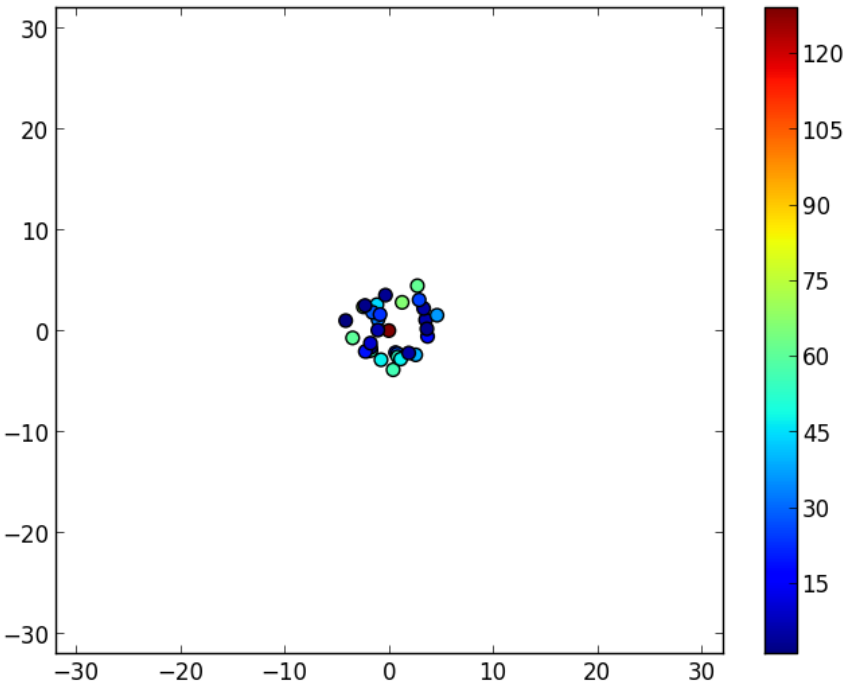


Figure 4-3: Probes At Step 575

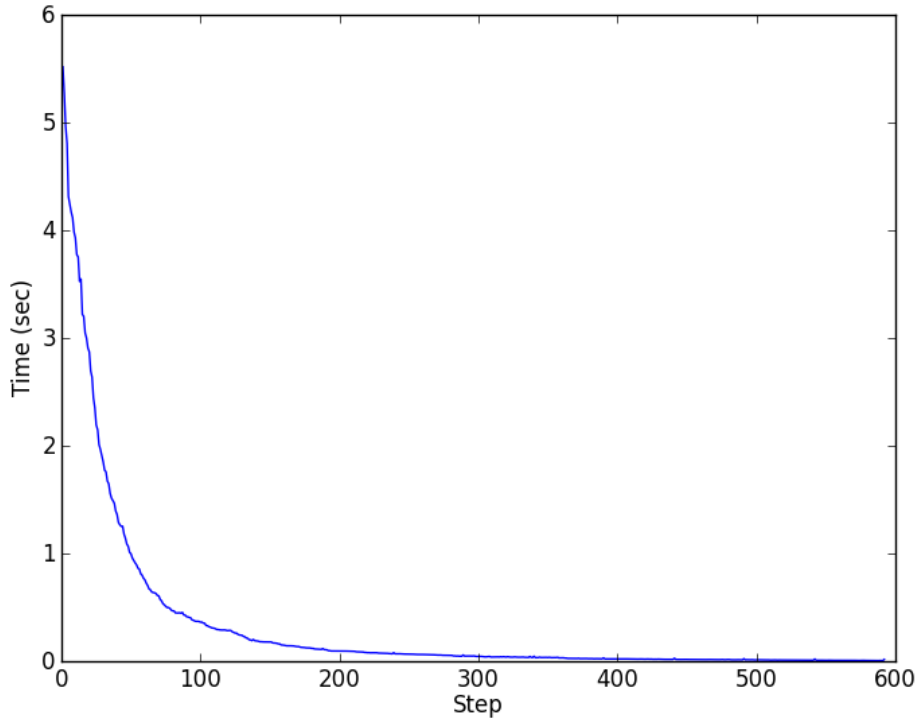


Figure 4-4: Multiplicity Factor Runtime Per Step For Test Function 01

## 4.6 Conclusions

When considering CFO runs where a maximum time is reached before the probes have converged, runtime for an implementation which includes a multiplicity factor may be longer. When exploring an unknown topology, is it worth it to take the risk of elongated runtime to include a multiplicity factor? Yes, the additional runtime added by the multiplicity factor is negligible compared to what it offers (less than 1% of the total runtime in every case). Of course, for runs where the maximum time is not reached, multiplicity factor combinations are guaranteed to occur as the probes converge and, thus, the runtime will always be shorter than without a multiplicity factor.

# Chapter 5

# Conclusions & Future Work

CFO's deterministic nature makes it a powerful algorithm and the advances described in this paper (and those referenced) make it even stronger. When implementing CFO, one should always use an array-based structure (Chapter 3) and implement a multiplicity factor (Chapter 4), each of which work to improve the algorithm's runtime. Still, there is much research that could be done to further improve CFO.

Different ranges of probe masses may greatly affect the behavior of a system of probes. It may be beneficial to compress the masses to a greater or lesser dynamic range using an indirect mass mapping. For instance, $2^{f(P_{\vec{p}})} \rightarrow P_m$, $\log f(x) \rightarrow \wp_m$, or constrain masses to the range $[0, 1]$ using probabilistic techniques. Since the resulting range of masses would be more uniform or more diverse, depending on the mapping technique used, and acceleration is directly dependent on probe mass differences, probes should exhibit acceleration which are more uniform or more diverse, again, depending on the mapping technique. This will enable the system to converge more slowly, to achieve finer detail in the resulting solution, or more quickly, to shorten runtime.

Optimal values for $\alpha$, $\beta$, and $\gamma$ have yet to be established. A link may exist between the objective function and these values, such as a proportional increase with the order

of the objective function. It might also happen that there is a set (or collection of sets) which is optimal for all objective functions. In either case, this work has yet to be completed and may hold important information about the movement of a system of probes that would shed light on further advancements, such as an optimal distribution of masses (as described in the preceding paragraph).

There exist cases where the probes resist convergence due to an acceleration property which is too high. That is, a probe (p) may overshoot a more optimal probe (q) because the influence from q to p is too large (q's mass greatly exceeds that of p). Probe p may then reverse its direction in the following step (again, due to q's influence, this time from the opposite direction), but once again overshoot q's position, returning to its original position. This can be referred to as *equal overshoot* and creates a cyclical behavior that, if not counteracted by neighboring probes, could be infinite. Thus, solutions between probes p and q are never properly evaluated due to this overshot. Disrupting these patterns could increase the rate of convergence.

# References


[1] J. K. S. Singh and R. Sinha, "A Comprehensive Survey on Various Evolutionary Algorithms on GPU," in *International Conference on Communication, Computing and Systems*, (Ferozepur, Punjab, India), August 2014.

[2] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[3] L. N. De Castro, *Fundamentals of natural computing.* Chapman & Hall/CRC, 2006.

[4] M. Mitchell, *An introduction to genetic algorithms.* MIT Press, 1996.

[5] L. N. De Castro, *Fundamentals of natural computing.* Chapman & Hall/CRC, 2006.

[6] J. Beasley and P. Chu, "A genetic algorithm for the set covering problem," *European Journal of Operational Research*, vol. 94, no. 2, pp. 392–404, 1996.

[7] D. J. Montana, "Strongly typed genetic programming," *Evolutionary Computation*, vol. 3, no. 2, pp. 199–230, 1995.

[8] J. R. Koza, *Genetic programming.* MIT Press, 1998.

[9] L. N. De Castro, *Fundamentals of natural computing.* Chapman & Hall/CRC, 2006.

[10] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *AIAA Journal*, vol. 41, no. 8, pp. 1583–1589, 2003.

[11] L. N. De Castro, *Fundamentals of natural computing.* Chapman & Hall/CRC, 2006.

[12] R. A. Formato, "Central Force Optimization: A New Nature Inspired Computational Framework for Multidimensional Search and Optimization," in *NICSO*, pp. 221–238, Springer-Verlag, 2007.

[13] R. A. Formato, "Central force optimisation: A new gradient-like metaheuristic for multidimensional search and optimisation," *Interational Journal of Bio-Inspired Computation*, vol. 1, no. 4, pp. 217–238, 2009.

[14] R. A. Formato, "Are Near Earth Objects the Key to Optimization Theory?," *Computing Research Repository*, December 2009.

[15] R. A. Formato, "Are Near Earth Objects the Key to Optimization Theory?," *British Journal of Mathematics & Computer Science*, vol. 3, pp. 341–351, April 2013.

[16] R. A. Formato, "Central Force Optimization: A New Deterministic Gradient-Like Optimization Metaheuristic," *Journal of the Operations Research Society of India*, vol. 46, no. 1, pp. 25–51, 2009.

[17] D. Ding, X. Luo, J. Chen, X. Wang, P. Du, and Y. Guo, "A Convergence Proof and Parameter Analysis of Central Force," *Journal of Convergence Information Technology*, vol. 6, no. 10, pp. 16–23, 2011.

[18] D. Ding, D. Qi, X. Luo, J. Chen, X. Wang, and P. Du, "Convergence analysis and performance of an extended central force optimization algorithm," *Applied Mathematics and Computation*, vol. 219, no. 4, pp. 2246–2259, 2012.

[19] R. A. Formato, "Central force optimization: A new metaheuristic with applica-

tions in applied electromagnetics," *Progress in Electromagnetics Research, PIER 77*, pp. 425–491, 2007.

[20] O. Roa, I. Amaya, F. Ramirez, and R. Correa, "Solution of nonlinear circuits with the central force optimization algorithm," in *IEEE 4th Colombian Workshop on Circuits and Systems*, (Barranquilla, Colombia), pp. 1–6, November 2012.

[21] G. Mohammad and N. Dib, "Synthesis of Antenna Arrays Using Central Force Optimization," in *Mosharaka International Conference on Communications, Computers and Applications*, (Amman, Jordan), 2009.

[22] G. Qubati, "Central force optimization method and its application to the design of antennas," Master's thesis, Jordan University of Science and Technology, 2009.

[23] G. M. Qubati and N. I. Dib, "Microstip Patch Antenna Optimization Using Modified Central Force Optimization," *Progress in Electromagnetics Research B*, vol. 21, pp. 281–298, 2010.

[24] G. Qubati, R. Formato, and N. Dib, "Antenna benchmark performance and array synthesis using central force optimisation," *Microwaves, Antennas Propagation, IET*, vol. 4, pp. 583–592, May 2010.

[25] K. R. Mahmoud, "Central Force Optimization: Nelder-Mead Hybrid Algorithm for Rectangular Microstrip Antenna Design," *Electromagnetics*, vol. 31, no. 8, pp. 578–592, 2011.

[26] A. Haghighi and H. M.Ramos, "Detection of Leakage Freshwater and Friction Factor Calibration in Drinking Networks Using Central Force Optimization," *Water Resources Management*, vol. 26, pp. 2347–2363, March 2012.

[27] N. Dib, A. Sharaqa, and R. A. Formato, "Variable $Z_0$ applied to Biogeography

Based Optimized Multi-Stub Matching Network and to a Central Force Optimized Meander Monopole," , April 2012. Submitted for Publication.

[28] R. A. Formato, "Central Force Optimization Applied to the PBM Suite of Antenna Benchmarks," *Computing Research Repository*, vol. abs/1003.0221, 2010.

[29] R. Green, L. Wang, and M. Alam, "Training neural networks using Central Force Optimization and Particle Swarm Optimization: Insights and comparisons," *Expert Systems with Applications*, vol. 39, pp. 555–563, January 2012.

[30] Y. Chen, J. Yu, Y. Mei, Y. Wang, and X. Su, "Modified central force optimization (mcfo) algorithm for 3d uav path planning," *Neurocomputing*, 2015.

[31] R. A. Formato, "Improved CFO Algorithm for Antenna Optimization," *Progress in Electromagnetics Research, PIER B*, vol. 19, pp. 405–425, 2010.

[32] R. A. Formato, "Parameter-Free Deterministic Global Search with Central Force Optimization," *Computing Research Repository*, vol. abs/1003.1039, 2010.

[33] R. A. Formato, "Central Force Optimization with variable initial probes and adaptive decision space," *Applied Mathematics and Computation*, vol. 217, no. 21, pp. 8866–8872, 2011.

[34] R. Green, L. Wang, M. Alam, and R. Formato, "Central Force Optimization on a GPU: A case study in high performance metaheuristics using multiple topologies," in *IEEE Congress on Evolutionary Computation*, (New Orleans, Los Angeles), pp. 550–557, June 2011.

[35] R. Green, L. Wang, M. Alam, and R. A. Formato, "Central force optimization on a GPU: A case study in high performance metaheuristics," *Journal of Supercomputing*, vol. 62, pp. 378–398, October 2012.

[36] R. S. Sinha and S. Singh, "Optimization Techniques on GPU: A Survey," in *International Multi Track Conference on Science, Engineering & Technical Innovations*, (Jalandar, India), June 2014.

[37] J. Liu and Y.-p. Wang, "An improved central force optimization algorithm for multimodal optimization," *Journal of Applied Mathematics*, vol. 2014, pp. 1–12, 2014.

[38] Y. Liu and P. Tian, "A multi-start central force optimization for global optimization," *Applied Soft Computing*, vol. 27, pp. 92–98, February 2015.

[39] W. Qian, B. Wang, and Z. Feng, "Adaptive central force optimization algorithm based on the stability analysis," *Mathematical Problems in Engineering*, vol. 2015, pp. 1–10, 2015.

[40] R. A. Formato, "Pseudorandomness in Central Force Optimization," *Computing Research Repository*, vol. abs/1001.0317, 2010.

[41] R. A. Formato, "Pseudorandomness in Central Force Optimization," *British Journal of Mathematics & Computer Science*, vol. 3, pp. 241–264, April 2013.

[42] R. A. Formato, "On the Utility of Directional Information for Repositioning Errant Probes in Central Force Optimization," *Computing Research Repository*, vol. abs/1005.5490, 2010.

[43] R. A. Formato, "Comparative Results: Group Search Optimizer and Central Force Optimization," *Computing Research Repository*, vol. abs/1002.2798, 2010.

[44] B. Xing and W.-J. Gao, "Central Force Optimization Algorithm," in *Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms*, vol. 62 of *Intelligent Systems Reference Library*, pp. 333–337, Springer International Publishing, 2014.

# Appendix A

# Test Configurations

| **IDP** | **dim** | **pD** | **nP** | **Time Limit** | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|---|---|---|---|
| UOA | 2 | 25 | 50 | 5000 | 1 | 2 | 0.5 |
| UOA | 2 | 25 | 50 | 5000 | 1 | 2 | 0.5 |
| UOA | 2 | 25 | 50 | 5000 | 1 | 2 | 0.5 |
| UOD | 2 | 50 | 50 | 5000 | 1 | 2 | 0.5 |
| UOD | 2 | 50 | 50 | 5000 | 1 | 2 | 0.5 |
| UOD | 2 | 50 | 50 | 5000 | 1 | 2 | 0.5 |
| Hyper | 2 | 7 | 49 | 5000 | 1 | 2 | 0.5 |
| Hyper | 2 | 7 | 49 | 5000 | 1 | 2 | 0.5 |
| Hyper | 2 | 7 | 49 | 5000 | 1 | 2 | 0.5 |

Table A.1: 2-D Test Configurations

| **IDP** | **dim** | **pD** | **nP** | **Time Limit** | $\alpha$ | $\beta$ | $\gamma$ |
|---|---|---|---|---|---|---|---|
| UOA | 2 | 25 | 50 | 1000 | 1 | 2 | 0.5 |
| UOA | 5 | 10 | 50 | 1000 | 1 | 2 | 0.5 |
| UOA | 10 | 5 | 50 | 1000 | 1 | 2 | 0.5 |
| UOD | 3 | 50 | 50 | 1000 | 1 | 2 | 0.5 |
| UOD | 5 | 50 | 50 | 1000 | 1 | 2 | 0.5 |
| UOD | 10 | 50 | 50 | 1000 | 1 | 2 | 0.5 |
| Hyper | 3 | 4 | 64 | 1000 | 1 | 2 | 0.5 |
| Hyper | 4 | 3 | 81 | 1000 | 1 | 2 | 0.5 |

Table A.2: n-D Test Configurations

# Appendix B

# Test Functions

| **Func** | $f(x_1, \cdots, x_n) =$ | $\Omega$ | **Min** | **Value** |
|---|---|---|---|---|
| 01 | $-20exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - exp(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)) + 20 + e$ | $[-32, 32]^n$ | (0,..., 0) | 0 |
| 02 | $(1.5 - x + xy)^2 + (2.25 - x + xy^2)^2 + (2.625 - x + xy^3)^2$ | $[-4.5, 4.5]^2$ | (3, 0.5) | 0 |
| 03 | $(x + 2y - 7)^2 + (2x + y - 5)^2$ | $[-10, 10]^2$ | (1, 3) | 0 |
| 04 | $100\sqrt{\lvert y - 0.01x^2\rvert} + 0.01\lvert x + 10\rvert$ | $[-3, 3]^2$ | (-10, 1) | 0 |
| 05 | $(4 - 2.1x^2 + \frac{x^4}{3})x^2 + xy + (4y^2 - 4)y^2$ | $[-3, 3]x[-2, 3]$ | $(0.0898, \mp 0.7126)$ | -1.0316 |
| 06 | $-cos(x)cos(y)e^{-((x-\pi)^2+(y-\pi)^2)}$ | $[-100, 100]^2$ | $(\pi, \pi)$ | -1 |
| 07 | $(1 + (x + y + 1)^2 \cdot (19 - 14x + 3x^2 - 14y + 6xy + 3y^2))\cdot$ $(30 + (2x - 3y)^2 \cdot (18 - 32x + 12x^2 + 48y - 36xy + 27y^2))$ | $[-2, 2]^2$ | (0, -1) | 0 |
| 08 | $1 + \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})$ | $[-512, 512]^n$ | (0,..., 0) | 0 |
| 09 | $sin^2(3\pi x) + (x - 1)^2(1 + sin^2(3\pi y)) + (y - 1)^2(1 + sin^2(2\pi y))$ | $[-10, 10]^2$ | (1, 1) | 0 |
| 10 | $0.26(x^2 + y^2) - 0.48xy$ | $[-10, 10]^2$ | (0, 0) | 0 |
| 11 | $\sum_{i=1}^{n}(\sum_{j=1}^{i}(x_j - j)^2)$ | $[-10.24, 10.24]^n$ | (1, 2, 3,..., n) | 0 |
| 12 | $\sum_{i=1}^{n} ix_i^4 + random[0, 1)$ | $[-1.28, 1.28]^n$ | (0,..., 0) | 0 + noise |
| 13 | $xsin(\sqrt{\lvert y + 1 - x\rvert})cos(\sqrt{\lvert x + y + 1\rvert}) +$ $(y + 1)cos(\sqrt{\lvert y + 1 - x\rvert})sin(\sqrt{\lvert x + y + 1\rvert})$ | $[512, 512]^2$ | (-488.6326, 512) | -511.73 |
| 14 | $10n + \sum_{i=1}^{n}(x_i^2 - 10cos(2\pi x_i))$ | $[-5.12, 5.12]^n$ | (0,..., 0) | 0 |
| 15 | $\sum_{i=1}^{n}(\sum_{j=1}^{i} x_j)^2$ | $[-64, 64]^n$ | (0,..., 0) | 0 |

| | | | | |
|---|---|---|---|---|
| 16 | $\sum_{i=1}^{n-1}(100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2)$ | $[-2.048, 2.048]^n$ | (1,..., 1) | 0 |
| 17 | $0.5 + \frac{sin^2(\sqrt{\sum_{i=1}^{n} x_i^2}) - 0.5}{[1+0.001 \sum_{i=1}^{n} x_i]^2}$ | $[-100, 100]^n$ | (0, ...,0) | 0 |
| 18 | $\sqrt{\sum_{i=1}^{n} x_i^2}$ | $[-100, 100]^n$ | (0,..., 0) | 0 |
| 19 | $\sum_{i=1}^{n}(-x_i sin(\sqrt{\lvert x_i \rvert})) + 418.982887n$ | $[-512, 512]^n$ | (420.968746,...,420.968746) | 0 |
| 20 | $max_i\{\lvert x_i \rvert, 1 \leq i \leq n\}$ | $[-100, 100]^n$ | (0,..., 0) | 0 |
| 21 | $\sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$ | $[-10, 10]^n$ | (0,..., 0) | 0 |
| 22 | $\sum_{i=1}^{n} x_i^2$ | $[-5.12, 5.12]^n$ | (0,..., 0) | 0 |
| 23 | $\sum_{i=1}^{n} \sum_{j=1}^{n} \frac{(100(x_i^2 - x_j)^2 + (1 - x_j)^2)^2}{4000}$ | $[-10.24, 10.24]^n$ | (1,...1) | 0 |

# Appendix C

# Python Object Statistics

**Function 01**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -51.05 | 6.25E-21 |
| 0.516 | 5.29E-03 | 0.75 | 2.90E-06 | -53.74 | 2.50E-21 |
| 0.562 | 1.67E-03 | 0.65 | 1.38E-04 | -84.71 | 7.14E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -56.31 | 1.08E-21 |
| 0.500 | 7.78E-03 | 0.52 | 4.37E-03 | -65.48 | 7.24E-23 |
| 0.617 | 3.40E-04 | 0.85 | 1.60E-08 | -8.98 | 4.57E-08 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -53.89 | 2.38E-21 |
| 0.621 | 3.02E-04 | 0.50 | 7.78E-03 | -60.25 | 3.22E-22 |

**Function 02**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -140.73 | 7.79E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -203.37 | 1.04E-31 |
| 0.547 | 2.49E-03 | 0.50 | 7.78E-03 | -190.23 | 3.44E-31 |

**Function 03**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -136.06 | 1.43E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -79.87 | 2.05E-24 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -178.66 | 1.06E-30 |

**Function 04**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -212.26 | 4.79E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -369.07 | 2.28E-36 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -443.62 | 8.30E-38 |

**Function 05**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -373.40 | 1.85E-36 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -307.86 | 5.95E-35 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -362.20 | 3.19E-36 |

**Function 06**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.53 | 3.87E-03 | -161.04 | 6.89E-30 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -206.63 | 7.77E-32 |
| 0.572 | 1.26E-03 | 0.50 | 7.78E-03 | -100.23 | 3.48E-26 |

**Function 07**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -128.12 | 4.21E-28 |
| 0.560 | 1.73E-03 | 0.51 | 6.87E-03 | -101.44 | 2.80E-26 |
| 0.505 | 6.90E-03 | 0.50 | 7.78E-03 | -143.60 | 5.42E-29 |

**Function 08**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -55.43 | 1.44E-21 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -119.70 | 1.43E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -232.76 | 9.12E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -197.32 | 1.78E-31 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -268.48 | 6.99E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -161.33 | 6.67E-30 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -214.14 | 4.09E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -324.09 | 2.36E-35 |

**Function 09**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -154.82 | 1.40E-29 |
| 0.553 | 2.11E-03 | 0.52 | 4.37E-03 | -114.97 | 2.95E-27 |
| 0.500 | 7.78E-03 | 0.54 | 3.14E-03 | -146.99 | 3.56E-29 |

**Function 10**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -3.12 | 5.95E-03 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -3.20 | 4.97E-03 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -24.53 | 2.76E-15 |

**Function 11**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -133.92 | 1.90E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -318.89 | 3.16E-35 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -204.49 | 9.37E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -179.45 | 9.83E-31 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -270.01 | 6.31E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -485.17 | 1.66E-38 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -321.37 | 2.75E-35 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -333.45 | 1.41E-35 |

**Function 12**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -123.41 | 8.27E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -276.69 | 4.06E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -15.98 | 4.45E-12 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -174.40 | 1.64E-30 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -260.06 | 1.24E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -253.67 | 1.94E-33 |

| | | | | | |
|---|---|---|---|---|---|
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -204.60 | 9.29E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -356.87 | 4.17E-36 |

**Function 13**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -170.74 | 2.41E-30 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | 2.86 | 1.04E-02 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -157.20 | 1.06E-29 |

**Function 14**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -151.48 | 2.07E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -207.10 | 7.46E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -4.89 | 1.18E-04 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -144.39 | 4.91E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -124.86 | 6.70E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -295.88 | 1.22E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -230.45 | 1.09E-32 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -10.43 | 4.66E-09 |

**Function 15**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -115.87 | 2.57E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | 0.76 | 4.57E-01 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -3.87 | 1.13E-03 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -138.71 | 1.01E-28 |

| | | | | | |
|---|---|---|---|---|---|
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -383.50 | 1.14E-36 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -287.84 | 2.00E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -148.06 | 3.12E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -436.07 | 1.13E-37 |

**Function 16**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -93.16 | 1.29E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -253.84 | 1.92E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -231.75 | 9.87E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -157.69 | 1.01E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -10.44 | 4.57E-09 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -404.54 | 4.37E-37 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -171.75 | 2.16E-30 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -348.48 | 6.40E-36 |

**Function 17**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -114.84 | 3.01E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -201.18 | 1.26E-31 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -417.47 | 2.48E-37 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -151.22 | 2.14E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -319.72 | 3.01E-35 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -319.71 | 3.02E-35 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -241.49 | 4.70E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -313.49 | 4.30E-35 |

**Function 18**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -99.61 | 3.89E-26 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -253.29 | 1.99E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -270.45 | 6.13E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -124.28 | 7.28E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -292.55 | 1.49E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -286.28 | 2.20E-34 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -253.45 | 1.97E-33 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -341.32 | 9.30E-36 |

**Function 19**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -8.39 | 1.24E-07 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -123.90 | 7.70E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -114.51 | 3.17E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -8.16 | 1.86E-07 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -16.14 | 3.74E-12 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -25.22 | 1.70E-15 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -33.18 | 1.35E-17 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -31.33 | 3.73E-17 |

**Function 20**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -95.48 | 8.32E-26 |

| | | | | | |
|---|---|---|---|---|---|
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -74.42 | 7.30E-24 |
| 0.514 | 5.61E-03 | 0.50 | 7.78E-03 | -71.02 | 1.69E-23 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -54.04 | 2.26E-21 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -87.77 | 3.78E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -112.44 | 4.41E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -87.75 | 3.79E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -95.73 | 7.94E-26 |

**Function 21**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -110.12 | 6.40E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -77.49 | 3.54E-24 |
| 0.619 | 3.19E-04 | 0.50 | 7.78E-03 | -96.19 | 7.28E-26 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -102.08 | 2.50E-26 |
| 0.500 | 7.78E-03 | 0.52 | 4.66E-03 | -92.24 | 1.55E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -148.22 | 3.06E-29 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -96.16 | 7.32E-26 |
| 0.500 | 7.78E-03 | 0.52 | 4.63E-03 | -134.68 | 1.72E-28 |

**Function 22**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -74.36 | 7.40E-24 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -138.71 | 1.01E-28 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -8.79 | 6.26E-08 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -96.05 | 7.48E-26 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -112.62 | 4.28E-27 |

| | | | | | |
|---|---|---|---|---|---|
| 0.807 | 1.53E-07 | 0.54 | 2.96E-03 | -90.69 | 2.10E-25 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -111.37 | 5.23E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -14.11 | 3.56E-11 |

**Function 23**

| Array | | Object Oriented | | T Test | |
|---|---|---|---|---|---|
| ks stat | p value | ks stat | p value | t stat | p value |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -61.97 | 1.95E-22 |
| 0.500 | 7.78E-03 | 0.56 | 1.99E-03 | -81.45 | 1.45E-24 |
| 0.621 | 3.01E-04 | 0.56 | 1.90E-03 | -75.38 | 5.80E-24 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -108.92 | 7.80E-27 |
| 0.500 | 7.78E-03 | 0.50 | 6.95E-03 | -15.41 | 8.23E-12 |
| 0.500 | 7.78E-03 | 0.51 | 5.96E-03 | -32.38 | 2.07E-17 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -108.22 | 8.76E-27 |
| 0.500 | 7.78E-03 | 0.50 | 7.78E-03 | -11.12 | 1.71E-09 |

# Appendix D

# Multiplicity Factor Statistics

**Function 01**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 155.89 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 98.97 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -1.32 | 0.20 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1440.08 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1464.50 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 2925.41 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 215.57 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 527.67 | 0.00 |

**Function 02**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 677.39 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 334.41 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 340.69 | 0.00 |

**Function 03**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 720.69 | 0.00 |
| 0.53 | 0.00 | 0.50 | 0.01 | 237.23 | 0.00 |
| 0.54 | 0.00 | 0.50 | 0.01 | 387.19 | 0.00 |

**Function 04**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | -21.98 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -77.97 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 269.81 | 0.00 |

**Function 05**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | -21.98 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -77.97 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 269.81 | 0.00 |

**Function 06**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 292.68 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 716.43 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 63.79 | 0.00 |

**Function 07**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.55 | 0.00 | 0.50 | 0.01 | -52.89 | 0.00 |
| 0.55 | 0.00 | 0.50 | 0.01 | 608.47 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 372.84 | 0.00 |

**Function 08**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 313.74 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 282.42 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -1.76 | 0.10 |
| 0.50 | 0.01 | 0.50 | 0.01 | 694.83 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1354.44 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 2423.06 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -93.50 | 0.00 |
| 0.50 | 0.01 | 0.51 | 0.01 | -205.01 | 0.00 |

**Function 09**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 375.82 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 705.76 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1086.18 | 0.00 |

**Function 10**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 43.27 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 37.58 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 175.08 | 0.00 |

**Function 11**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 328.14 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 142.72 | 0.00 |
| 0.60 | 0.00 | 0.51 | 0.01 | 456.92 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1659.42 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 2212.73 | 0.00 |
| 0.56 | 0.00 | 0.50 | 0.01 | 866.21 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1291.38 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 226.70 | 0.00 |

**Function 12**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 412.08 | 0.00 |
| 0.50 | 0.01 | 0.73 | 0.00 | 48.03 | 0.00 |
| 0.50 | 0.01 | 0.62 | 0.00 | 26.17 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 2166.28 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1637.44 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1436.53 | 0.00 |

| | | | | | |
|---|---|---|---|---|---|
| 0.50 | 0.01 | 0.50 | 0.01 | 483.86 | 0.00 |
| 0.50 | 0.01 | 0.79 | 0.00 | 53.39 | 0.00 |

**Function 13**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 898.18 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1058.28 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 393.67 | 0.00 |

**Function 14**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | -59.56 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 62.27 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.71 | 0.49 |
| 0.50 | 0.01 | 0.50 | 0.01 | 816.09 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 843.73 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1003.35 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -217.09 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -10.29 | 0.00 |

**Function 15**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 1116.53 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -4.00 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -3.97 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 551.31 | 0.00 |

| | | | | | |
|---|---|---|---|---|---|
| 0.50 | 0.01 | 0.50 | 0.01 | 293.97 | 0.00 |
| 0.50 | 0.01 | 0.56 | 0.00 | -42.70 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 138.36 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -15.65 | 0.00 |

**Function 16**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 180.58 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 149.00 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.58 | 0.57 |
| 0.50 | 0.01 | 0.50 | 0.01 | 706.91 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 301.59 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 494.92 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 860.56 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 3874.51 | 0.00 |

**Function 17**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 137.77 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 66.75 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.57 | 0.58 |
| 0.50 | 0.01 | 0.50 | 0.01 | -31.41 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -144.73 | 0.00 |
| 0.53 | 0.00 | 0.50 | 0.01 | -22.24 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -71.36 | 0.00 |
| 0.54 | 0.00 | 0.50 | 0.01 | -84.40 | 0.00 |

**Function 18**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 1493.72 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 662.65 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -1.21 | 0.24 |
| 0.50 | 0.01 | 0.50 | 0.01 | 492.23 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 769.24 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 1385.53 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 24.79 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 14.22 | 0.00 |

**Function 19**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 468.92 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 386.15 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 734.71 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -6.04 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 51.77 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 67.60 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 347.89 | 0.00 |
| 0.50 | 0.01 | 0.57 | 0.01 | 65.69 | 0.00 |

**Function 20**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 62.69 | 0.00 |

| | | | | | |
|---|---|---|---|---|---|
| 0.50 | 0.01 | 0.50 | 0.01 | 758.50 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.22 | 0.83 |
| 0.50 | 0.01 | 0.50 | 0.01 | 3.44 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 4.87 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 7.69 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.45 | 0.66 |
| 0.50 | 0.01 | 0.50 | 0.01 | -41.25 | 0.00 |

**Function 21**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 849.50 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 73.55 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.07 | 0.95 |
| 0.50 | 0.01 | 0.50 | 0.01 | 610.91 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 788.06 | 0.00 |
| 0.52 | 0.00 | 0.50 | 0.01 | 134.47 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 444.80 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -13.47 | 0.00 |

**Function 22**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | 849.50 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 73.55 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -0.07 | 0.95 |
| 0.50 | 0.01 | 0.50 | 0.01 | 610.91 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 788.06 | 0.00 |

| | | | | | |
|---|---|---|---|---|---|
| 0.52 | 0.00 | 0.50 | 0.01 | 134.47 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 444.80 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -13.47 | 0.00 |

**Function 23**

| Original | | Multiplicity Factor | | T Test | |
|---|---|---|---|---|---|
| KS stat | P value | KS stat | P value | T stat | P value |
| 0.50 | 0.01 | 0.50 | 0.01 | -35.83 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 38.26 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -1.08 | 0.29 |
| 0.50 | 0.01 | 0.50 | 0.01 | 173.97 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 308.29 | 0.00 |
| 0.50 | 0.01 | 0.60 | 0.00 | 110.49 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | 671.55 | 0.00 |
| 0.50 | 0.01 | 0.50 | 0.01 | -16.15 | 0.00 |