Global Optimization of Pairwise Comparisons Matrix Based on Differential Evolution Algorithm

by

Yuqing Duan

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science (MSc) in Computational Sciences

The Faculty of Graduate Studies
Laurentian University
Sudbury, Ontario, Canada

THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE
Laurentian Université/Université Laurentienne
Faculty of Graduate Studies/Faculté des études supérieures

Title of Thesis Titre de la thèse	Global Optimization of Pairwise Comparisons Matrix Based on Differential Evolution Algorithm	
Name of Candidate Nom du candidat	Duan, Yuqing	
Degree Diplôme	Master of Science	
Department/Program Département/Programme	Computational Sciences	Date of Defence Date de la soutenance July 22, 2019

APPROVED/APPROUVÉ

Thesis Examiners/Examinateurs de thèse:

Dr. Waldermar W. Koczkodaj
(Supervisor/Directeur de thèse)

Dr. Miroslaw Mazurek
(Committee member/Membre du comité)

Dr. Kalp Passi
(Committee member/Membre du comité)

Dr. Grzegorz Marcin Wójcik
(External Examiner/Examinateur externe)

Approved for the Faculty of Graduate Studies
Approuvé pour la Faculté des études supérieures
Dr. David Lesbarrères
Monsieur David Lesbarrères
Dean, Faculty of Graduate Studies
Doyen, Faculté des études supérieures

ACCESSIBILITY CLAUSE AND PERMISSION TO USE

Abstract

Pairwise comparisons matrices (PCM) are commonly used when different entities or abstract concepts are compared for making decisions. The compared entities for decision making can be both subjective and objective indicators. The elements in PCM are ratios in case of multiplicative version. Using differential evolution (DE) heuristic algorithm, we can find an optimal solution for a given PCM. The optimization results are fairly good with geometric mean (GM) and eigenvector (EV). The thesis provides an introduction of differential evolution and how to apply it to the global optimization of PC matrices. IDEs and Java/R packages used in the Monte Carlo experiment will be discussed. Some results of considerable importance for PC matrices that have been obtained will also be illustrated in the thesis.

Keywords

Pairwise comparisons matrices, Differential Evolution heuristic, Optimization, Geometric mean, Eigenvector, Monte Carlo experiment, Java, R.

Table of Contents

List of Figures

1 Introduction of Genetic Algorithm

1.1 Overview of GA Theory

Genetic Algorithm (GA) is a computational model to simulate the natural selection and genetic mechanism of Darwin's biological evolution theory [31]. It is a method to search for the optimal solution by simulating the natural evolution process. The genetic algorithm begins with a population representing the potential solution set of the problem. The population consists of a certain number of individuals encoded by genes. Each individual is a phenotype of the genotype.

As the main carrier of genetic material, the chromosome is the collection of multiple genes. Its internal expression (i.e., genotype) is a kind of genome combination. It determines the external expression of individual's shape. For example, the characteristics of black hair are determined by some genome combination which controls this feature in a chromosome. Therefore, the mapping from phenotype to genotype, the coding, is needed at the beginning. Because the work of copying gene coding is very complex, we often simplify it, such as utilizing binary coding. Generations evolve to produce better and better approximate solutions after the initialization of the first generation population, according to the rule "survival of the fittest." In each generation, individuals are selected according to the fitness value of the individual in the problem domain.

The genetic operators of genetics carry out crossover and mutation to

produce a population representing a new set of solutions. This process will lead to a population like a natural evolution that is more adaptable to the environment than its predecessors. The decoding of the optimal individuals in the last generation can be used as the approximate optimal solution of the problem [17].

1.2 General Operation Process of GA

Genetic algorithm (GA) is a kind of randomized search method, which is based on the evolutionary law of biology (survival of the fittest genetic mechanism). It was first proposed by Professor J. Holland of the United States in 1975.

Its main characteristics are that it operates directly on structural objects without the restriction of derivative and function continuity; it has inherent, implicit parallelism and better global optimization ability; it can automatically obtain and guide the optimization search space by using probabilistic optimization method, adjust the search direction adaptively, and do not need definite rules. Then, these properties of genetic algorithm have been widely used in combinatorial optimization, machine learning, signal processing, adaptive control, and artificial life. It is a key technology in modern intelligent computing.

For an optimization problem of finding the maximum value of a function (the minimum value of a function is the same), it can generally be described as the following mathematical programming model:

$$\begin{cases} \max f(X) \\ X \in R \\ R \subset U \end{cases}$$

Figure 1: A Model for Finding the Maximum Value of $f(x)$

For the formula in Figure 1, X is the decision variable, $maxf(x)$ is the objective function, $X \in R$ and $R \in U$ is the constraint condition, U is the basic space, R is a subset of U. Solutions satisfying constraints are called feasible solutions and set R represents the set of all solutions satisfying constraints, which is called a feasible solution set.

Genetic algorithm is also a search heuristic algorithm used to solve optimization in the field of computer science and artificial intelligence [20]. It is a kind of evolutionary algorithm. This heuristic is often used to generate useful solutions to optimize and search for problems. Evolutionary algorithms originally developed from some phenomena in evolutionary biologies, such as heredity, mutation, natural selection, and hybridization. In the case of improper selection of fitness function, the genetic algorithm may converge to local optimum, but not to the global optimum. The primary operation procedure of genetic algorithm is as follows:

- Initialization:

  Set the evolutionary algebra counter $t = 0$, set the maximum evolutionary algebra T, and randomly generate M individuals as the initial population $P(0)$.

- Individual evaluation:

  The fitness value of each individual in $P(t)$ was calculated.

- Selection operation:

  The selection operator is applied to the group. The purpose of selection is to inherit the optimized individuals directly to the next generation or to produce new individuals through paired crossover and then to the next generation. Selection is based on the fitness evaluation of individuals in a group.

- Crossover operation:

  The crossover operator is applied to the population. The crossover operator plays a key role in genetic algorithm.

- Mutation operation:

  The mutation operator is applied to the population. That is, to change the gene value at some loci of individual strands in a population. Population $P(t)$ is selected, crossed and mutated to obtain the next generation population $P(t+1)$.

- Termination condition judgment:

  If $t = T$, the calculation is terminated by taking the individual with the maximum fitness obtained in the evolution process as the output of the optimal solution.

1.3 Characteristics of GA

Genetic algorithm is a general algorithm to solve the problem of search, and it can be used for various general problems. The common characteristics of these search algorithms are:

- A set of candidate solutions is formed.
- Calculate the fitness of these candidate solutions according to some adaptive conditions.
- Reserve some candidate solutions according to fitness, and abandon other candidate solutions.
- Do some operations on reserved candidate solutions to generate new candidate solutions.

In a genetic algorithm, these features are combined in a special way: parallel search based on chromosome group, selection operation with guessing nature, exchange operation and mutation operation. This special combination method distinguishes the genetic algorithm from other search algorithms.

In addition to the above characteristics, the genetic algorithm also has the following characteristics:

- Genetic algorithm starts from the string a set of initial solution, rather than starting from a single solution. This is a great difference between genetic algorithm and traditional optimization algorithm. The

traditional optimization algorithm is to get the optimal solution from a single initial value iteration, and it is easy to get into the optimal local solution. The genetic algorithm starts from a string set and has large coverage, which is favorable for global optimization.

- Genetic algorithm deals with multiple individuals in a group at the same time, that is, evaluating multiple solutions in the search space, which reduces the risk of falling into optimal local solution, and the algorithm itself is easy to realize parallelization.

- Genetic algorithm does not need the knowledge of search space or other auxiliary information, but only uses fitness function value to evaluate the individual, and carries on the genetic operation on this basis. The fitness function is not only constrained by continuous differentiability but also can be set arbitrarily. This feature greatly extends the application scope of the genetic algorithm.

- Genetic algorithm does not use deterministic rules but uses probabilistic transition rules to guide its search direction.

- It has self-organizing, self-adaptive and self-learning habits. When genetic algorithm organizes and searches by itself using the information obtained from the evolution process, the individuals with higher fitness have higher survival probability and get a more adaptive genetic structure.

- In addition, the algorithm itself can also adopt dynamic adaptive technology to automatically adjust the algorithm control parameters and coding accuracy during the evolution process, such as the use of the fuzzy adaptive method.

1.4 Basic Framework of GA

1.4.1 Coding

Genetic algorithms can not directly deal with the parameters of the problem space, and parameters must be transformed into genetic space by the gene according to a specific structure of chromosomes or individuals. This transformation is called encoding, or representation.

The following three criteria are often used to evaluate coding strategies:

- Completeness: all points (candidate solutions) in the problem space can be represented as points (chromosomes) in the GA space.
- Soundness: chromosomes in GA space correspond to candidate solutions in all problem spaces.
- Non-redundancy: chromosomes and candidate solutions are a one-to-one correspondence.

At present, several commonly used encoding technologies are binary encoding, floating-point encoding, character encoding, coding and so on.

Binary coding is the most commonly used encoding method in genetic algorithm. That is, the binary character set {0, 1} produces the usual 01 string to represent the candidate solution in the problem space. It has the following characteristics:

- Simple and easy to use.
- Conforming to the minimum character set coding principle.
- It is easy to analyze with schema theorem because schema theorem is based.

1.4.2 Fitness Function

Adaptability in evolutionism is the ability of an individual to adapt to the environment and to reproduce offspring. The fitness function of the genetic algorithm is also called the evaluation function, which is used to judge the individual's fitness in the group. It is evaluated according to the objective function of the problem [32].

Generally, the genetic algorithm does not need other external information in the process of searching evolution, only use evaluation function to evaluate the individual or solution, and as a basis for future genetic operation. In the genetic algorithm, the fitness function is sorted by comparison, and the selection probability is calculated on this basis, so the value of the fitness function should be positive. Therefore, in many cases, it is necessary to map

the objective function to the fitness function with the maximum value and non-negative function value.

The design of fitness function mainly satisfies the following conditions:

- Single value, continuous, non negative and maximization.
- Reasonable and consistent.
- Small amount of calculation.
- Strong generality.

In specific applications, the design of fitness function depends on the requirements of solving the problem itself. Fitness function design directly affects the performance of the genetic algorithm.

1.4.3 Selection of Initial Population

In the genetic algorithm, the individuals in the initial group are randomly generated. Generally speaking, the initial group setting can take the following strategies:

- According to the inherent knowledge of the problem, try to grasp the distribution range of the space occupied by the optimal solution in the whole problem space, and then set the initial group in this distribution range.

- First, a certain number of individuals are randomly generated, and then the best individuals are selected to add to the initial group. This process is iterated until the number of individuals in the initial population reaches a predetermined scale [40].

1.5 General Methods in GA

1.5.1 Initial State Establishment

Initial populations are randomly selected from solutions, which are likened to chromosomes or genes. This population is called the first generation, unlike symbolic AI systems where the initial state of the problem is given.

1.5.2 Fitness Assessment

Each solution (chromosome) is assigned a fitness value, which is specified according to the actual proximity of the solution (in order to approximate the solution). Don't confuse these "solutions" with the "answers" to a question, but understand them as the characteristics that the system may need to take advantage of in order to get an answer.

1.5.3 Reproduction (including offspring mutation)

Those chromosomes with higher fitness values are more likely to produce offspring (which will mutate as well). The offspring are the offspring of the parents, who combine genes from their parents, a process known as hybridiza-

tion.

If a new generation contains a solution that produces an output sufficiently close to or equal to the expected answer, the problem is solved. If that's not the case, the new generation will repeat the reproductive process their parents did and evolve from generation to generation until the desired solution is reached.

1.6 Terminology in GA

- Individuals: chromosomes can also be called individuals. A certain number of individuals form a population group. The number of individuals in a population is called population size.

- Genes: genes are elements in strings, and genes are used to represent individual characteristics. For example, if there is a string S=1011, 1, 0, 1, 1 of these four elements are called genes.

- Gene Position: gene loci represent the position of a gene in a string in an algorithm called Gene Position, sometimes referred to as a gene locus. The gene position is calculated from the left to the right of the string, for example, in the string S=1101, the gene position of 0 is 3.

- Eigenvalue: the eigenvalue of a gene is the same as the weight of a binary number when using string to represent integers; for example, in string S = 1011, 1 in gene position 3 has a gene eigenvalue of 2; 1 in gene position 1 has a gene eigenvalue of 8.

- Fitness: the adaptation degree of each individual to the environment is called fitness. In order to reflect the adaptability of chromosomes, the fitness function, which can measure every chromosome in the problem, is introduced. This function is used to calculate the probability of individuals being used in groups.

1.7 Deficiencies of GA

- The coding is not standardized, and the representation is inaccurate.
- Single genetic algorithm coding can not fully express the constraints of optimization problems. One way to consider constraints is to use thresholds for infeasible solutions so that the computational time will inevitably increase.
- The efficiency of genetic algorithm is usually lower than other traditional optimization methods [23].
- Genetic algorithm is easy to converge prematurely.
- There is no effective quantitative analysis method for the accuracy, feasibility and computational complexity of the genetic algorithm.

2 Differential Evolution Heuristic

2.1 Relationship between GA and DE

Differential evolution (DE) is an efficient and powerful population-based stochastic search technique for solving optimization problems over continuous space, which has been widely applied in many scientific areas and engineering fields [36]. Differential evolution algorithm is similar to the genetic algorithm. It is also an optimization algorithm based on modern intelligence theory, which guides the direction of optimization search by the swarm intelligence produced by the cooperation and competition among individuals in a group.

2.2 Introduction of Differential Evolution Heuristic

Differential evolution (DE) is a search heuristic introduced by Storn and Price (1997). Its remarkable performance as a global optimization algorithm on continuous numerical minimization problems has been extensively explored; see Price et al. (2006). DE belongs to the class of genetic algorithms which use biology-inspired operations of crossover, mutation, and selection on a population in order to minimize an objective function throughout successive generations (see Mitchell, 1998). As with other evolutionary algorithms, DE solves optimization problems by evolving a population of candidate solutions using alteration and selection operators. DE uses floating-point instead of bit-string encoding of population members, and arithmetic operations instead

of logical operations in mutation. DE is particularly well-suited to find the global optimum of a real-valued function of real-valued parameters and does not require that the function be either continuous or differentiable [33].

Differential evolution algorithm (DE) is an efficient global optimization algorithm. It is also a group-based heuristic search algorithm. Each individual in the group corresponds to a solution vector. The evolutionary process of differential evolution algorithm is very similar to that of genetic algorithm, which includes mutation, crossover and selection operations, but the definitions of these operations are different from those of genetic algorithm.

It is mainly used to solve real optimization problems. This algorithm is a group-based adaptive global optimization algorithm, which belongs to the evolutionary algorithm. Because of its simple structure, easy implementation, fast convergence, and strong robustness, it is widely used in data mining, pattern recognition, digital filter design, artificial neural network, electromagnetics, and other fields. In the first International Evolutionary Computing (ICEO) Competition held in Nagoya, Japan in 1996, Differential Evolutionary Algorithms (DEA) proved to be the fastest evolutionary algorithm.

Like genetic algorithm, Differential Evolution (DE) is an optimization algorithm based on modern intelligence theory, which guides the direction of optimization search through the swarm intelligence generated by the cooperation and competition among individuals within a group. The basic idea of this algorithm is: starting from a random initial population, a new individual

is generated by summing the vector difference between any two individuals in the population and the third individual, and then comparing the new individual with the corresponding individual in the contemporary population. If the fitness of the new individual is better than that of the current individual, the old individual will be replaced by the new individual in the next generation, otherwise. Still preserving the old individual. Through continuous evolution, good individuals are retained, inferior individuals are eliminated, and search is guided to approach the optimal solution.

In order to make more researchers understand and study the differential evolution algorithm, Storn and Price established the official website of the differential evolution algorithm in 1997. The establishment of the website has attracted the attention and support of the majority of researchers, which provides a great convenience for the relevant personnel to study the theory and application of the differential evolution algorithm. In addition, Store and Price have not applied for any kind of patent on differential evolution algorithm, which also plays an important role in promoting the research and application of the differential evolution algorithm.

DE algorithm generates population individuals by coding with floating-point vectors. In the process of optimizing DE algorithm: firstly, two individuals are selected from the parent to generate difference vectors; secondly, another individual and difference vectors are selected to sum and generate experimental individuals; secondly, the parent and corresponding experimental individuals are cross-operated to generate new offspring individuals; finally,

the selection operation is carried out between the parent and the offspring individuals. To preserve eligible individuals in the next generation.

2.3 General Steps in Differential Evolution

Firstly, two individuals are selected from the parent to make difference to generate difference vectors; Secondly, another individual and difference vectors are selected to sum and generate experiment individuals; Thirdly, the parent and corresponding experiment individuals are crossed to generate new generation individuals; Finally, the father is selected to generate a new generation of individuals. Selection between generations and offspring is performed to preserve eligible individuals in the next generation.

The main control parameters of the DE algorithm include population size (NP), scaling factor (F) and crossover probability (CR).

NP mainly reflects the size of population information in the algorithm. The larger the NP value, the richer the population information contains. However, the consequence is that the calculation amount becomes larger, which is not conducive to solving. On the contrary, the diversity of the population is limited, which is not conducive to the global optimization of the algorithm, and even leads to search stagnation [35].

CR mainly reflects the degree of information exchange between offspring, parents and intermediate variants in the process of crossover. The greater the value of CR, the greater the degree of information exchange. Conversely, if the value of CR is small, the diversity of the population will decrease rapidly,

which is not conducive to global optimization.

Compared with CR, F has a greater impact on the performance of the algorithm, and F mainly affects the global optimization ability of the algorithm. The smaller the F, the better the local searchability of the algorithm [8]. The larger the F, the more the algorithm can jump out of the local minimum, but the convergence speed will slow down. In addition, F also affects population diversity.

2.3.1 Initialization

DE uses some real-valued parameters of n-dimensions as the population of each generation, and an individual can be represented as:

$$X_{i,G}(i = 1, 2, \cdots, NP) \tag{1}$$

In the formula(1), i is the serial number of the individual in the population; G is the serial number of generation; NP is the population size, and NP remains unchanged during whole evolution. A In order to establish the initial point of optimal search, the population must be initialized. Usually, one way to find an initial population is to choose randomly from the values in a given boundary constraint. In DE research, it is generally assumed that all randomly initialized populations conform to a uniform probability distribution.

$$X_{ji,0} = rand(0, 1)(X_j^{(U)} - X_j^{(L)}) + X_j^{(L)} \tag{2}$$

$$(i = 1, 2, \cdots, NP; j = 1, 3, \cdots, D)$$

In formula(2): rand[0, 1] - the uniform random number generated between 0 and 1. If the initial solution of the problem can be obtained in advance, the initial population can also be generated by adding random deviation of the normal distribution to the initial solution, which can improve the reconstruction effect.

2.3.2 Mutation

For each target vector, $X_{i,G}(i = 1, 2, \cdots, NP)$, the variant vectors are generated as follows:

$$X_{i,G+1} = (X_{r_1,G} + F(X_{r_2,G} - X_{r_3,G}) \tag{3}$$

In formula(3), the randomly selected sequence numbers r_1, r_2 and r_3 are different from each other. r_1, r_2 and r_3 are also different from the target vector sequence number i, so it must satisfy $NP \geq 4$. The mutation operator $F \in [0, 2]$ is a real constant factor that controls the amplification of deviation variables.

2.3.3 Crossover

In order to increase the diversity of interference parameter vectors, crossover operation is needed. Therefore, the test vector becomes:

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \cdots, u_{Di,G+1}) \tag{4}$$

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if rand } b(j) \leq CR \text{ or } j = rnbr(i) \\ X_{ji,G+1}, & \text{if rand } b(j) \geq CR \text{ or } j \neq rnbr(i) \end{cases} \tag{5}$$

$$(i = 1, 2, \cdots, NP; j = 1, 3, \cdots, D)$$

In formula(5), $randb(j)$ produces the j-th estimate of the random number generator between [0,1]; rnbr(i) = 1,2,..., D is a selected sequence, it is used to ensure at least one parameter; CR is crossover operator, the range is [0,1].

2.3.4 Selection

To determine whether the test vector $u_{i,G+1}$ will become a member of the next generation, DE compares the test vector with the target vector in the current population according to the greedy criterion.

If the objective function is to be minimized, then the vector with smaller objective function values will win a place in the next generation. All the individuals in the next generation are better or at least as good as the corresponding individuals in the current population. DE's selection criterion demands that improved vectors always be accepted [34].

Note that the test vector in the DE selector is only compared to one individual, not to all individuals in the current population.

2.3.5 Boundary Condition Processing

In the problem with boundary constraints, it is necessary to ensure that the parameter values of the new individuals are located in the feasible region of the problem.

A simple method is to replace the new individuals which do not conform to the boundary constraints with the parameter vectors randomly generated in the feasible region. That is: if $u_{ji,G+1} \leq x_j^{(L)}$, then:

$$u_{ji,G+1} = randj[0,1](X_j^{(U)} - X_j^{(L)}) + X_j^{(L)} \tag{6}$$

Another method is to reproduce the test vectors according to formula(6) and then do crossover operation until the new individuals satisfy the boundary constraints, but this is inefficient.

2.4 Flowchart of Differential Evolution Heuristic

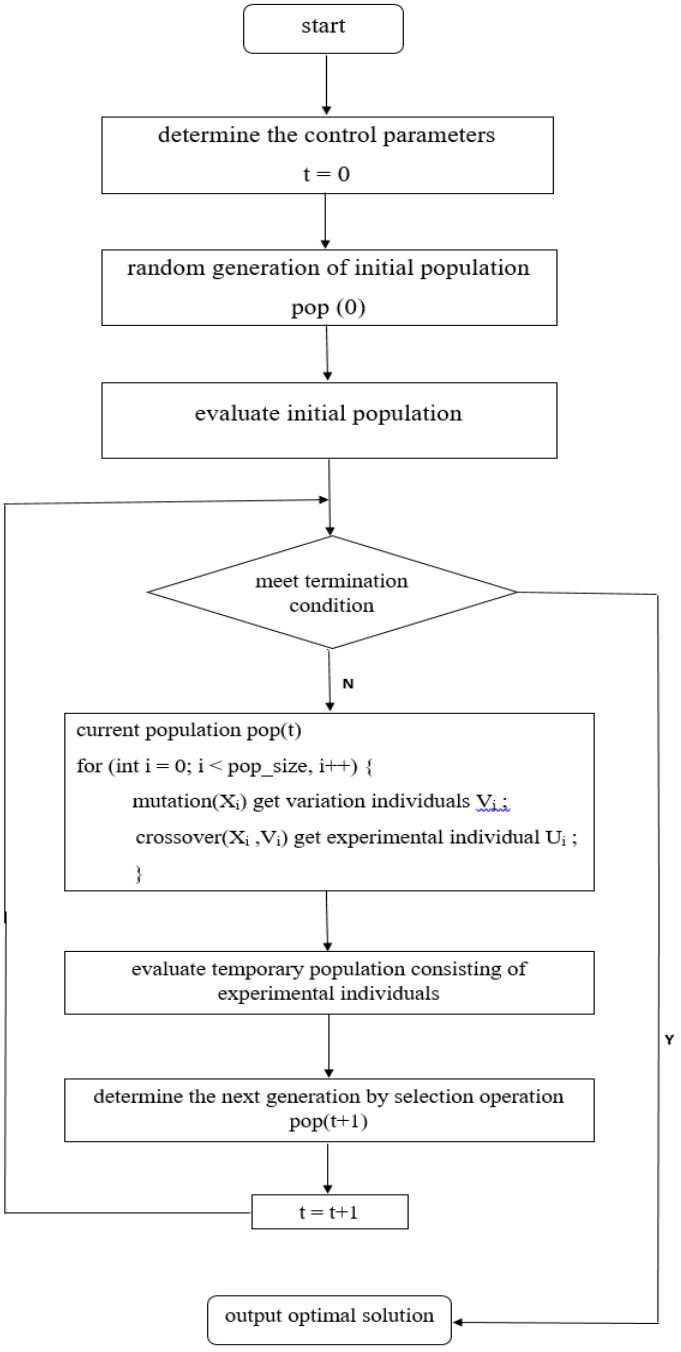


Figure 2: Flowchart of Differential Evolution Heuristic

2.5 UML Class Diagram

In order to clearly express the structure of the program in the appendix (section 13.1.1) and the calling relationship between different java classes, I use a UML class diagram to describe the whole project.

2.5.1 A Brief Introduction of UML Class Diagram

The class diagram is a graph describing classes, interfaces, collaborations, and their relationships. It is used to display the static structure of each class in the system. Class diagrams are the basis for defining other diagrams. Based on class diagrams, state diagrams, collaboration diagrams, component diagrams, and configuration diagrams can be used to describe the characteristics of other aspects of the system further.

Class diagrams include seven elements: class, interface, collaboration, dependency, generalization, association, and realization. The four essential element modules in UML class diagram will be introduced separately in the following sections.

2.5.2 Class

A class defines a set of objects with state and behavior. Attributes and associations are used to describe states. Attributes are usually represented by unidentified data values, such as numbers and strings. The relationship between identifiable objects represents association. Behavior is described by operation, and the method is the realization of operation. The lifetime of

the object is described by the state machine attached to the class.

1. Name: The name of a class is all the constituent elements of each class.

2. Attribute: (1) Visibility: The visibility of attributes in classes mainly includes public, private and protected attributes. In UML, the public type is expressed in "+", the private type in "-", and the protected type in "#". There is no default visibility in UML classes, and if no symbols are displayed, the visibility of the attribute is not defined. (2) Attribute name: According to the UML convention, the single-character attribute name is lowercase. If the attribute name contains more than one word, these words are merged, and the initial letters of all words except the first one are capitalized. (3) Attribute string. Attribute strings are used to specify other information about attributes, such as an attribute that should be permanent. Any rule that you want to add to the attribute definition string value but there is no suitable place to add it can be placed in the attribute string. (4) Class attributes. Attributes can also be defined as a class attribute, which means that all objects of the class share the attribute. In class diagrams, class attributes are underlined.

3. Operation. The operation of a class is an abstraction of what an object of a class can do, which is equivalent to the implementation of a service.

4. Responsibilities: In the area below the operation section, you can use it to illustrate the responsibilities of the class. Responsibility is the contract or obligation of a class or other element. The duty of a class

is to write a free-form text, a phrase, a sentence, etc. In UML, the responsibilities are listed in the partition bar at the bottom of the class diagram.

5. Constraint. Explaining the responsibility of a class is an informal way to eliminate ambiguity, and the formal way is to use constraints. Constraints specify one or more rules to be satisfied by this class. In UML, constraints are free text enclosed in curly brackets.

2.5.3 Interface

Interfaces contain operations but do not contain attributes, and they do not have visible associations with the outside world.

2.5.4 Relationships between Classes

There are four most common relationships among classes: dependency, generalization, association, and realization.

1. Dependency: Dependency represents the semantic relationship between two or more model elements. It represents a situation where specific changes to an element (provider) may affect or provide messages to other elements (customers), that is, customers depend in some form on other class elements. According to this definition, associations, implementations, and generalizations are dependencies, but they have more specific semantics. In UML, depending on a virtual arrow from the

customer to the provider, a constructed keyword is used to distinguish its type. UML defines four basic dependency types: Usage dependency, Abstraction dependency, Permission dependency, and Binding dependency. (1) Use dependence. Use dependency is very straightforward, usually indicating that the customer uses the service provided by the provider to implement its behavior. (2) Abstract dependency. Abstract dependency is used to represent the relationship between customers and providers and depends on things at different levels of abstraction. (3) Authorization dependence. Authorization dependency represents the ability of one thing to access another. Providers can control and restrict access to their content by specifying the rights of customers. (4) Binding dependencies. Binding dependencies are higher-level dependency types used to bind templates to create new model elements.

2. Generalization: Generalization relation is a kind of classification relation existing between general elements and specific elements. It is only used on type, not instance. In classes, general elements are called superclasses or superclasses, while specific elements are called subclasses. In UML, a generalization relationship is represented by a hollow triangular arrow pointing from a subclass to a parent class.

3. Association: Association relation is a structural relationship, which indicates the relationship between the object of one thing and the object of another thing. In other words, associations describe discrete connec-

tions between objects or instances in a system. In UML, associations are represented by a solid line connecting two classes. There are six corresponding modifications of association: name, role, multiplicity, aggregation, composition, and navigation. (1) Name. Names are used to describe the nature of associations. Usually, a verb or verb phrase is used to name associations. Names are prefixed or suffixed with a direction indicator that guides reading to eliminate possible ambiguities in the meaning of names, which are represented by a solid triangular arrow. (2) Role. Roles are the responsibilities of one class to another in a relationship. Role names are nouns or noun phrases to explain how objects participate in the association. (3) Multiplicity. Constraints are one of the three extensions of UML, and multiplicity is one of the most widely used constraints. The multiplicity of association refers to how many objects can participate in the association. Multiplicity can be used to express a range of values, a specific value, an infinite range or a set of discrete values. (4) Aggregation. Aggregated relationships represent relationships between whole and part relationships. The aggregation relationship describes the relationship of "has a". In UML, aggregation relationships are represented by solid lines with holes where the head points to the whole. (5) Composition. Combination relationship is a particular case of aggregation relationship, which is a stronger form of aggregation, also known as secure aggregation. In composition, the life cycle of member objects depends on the life cycle

of aggregation, which not only controls the behavior of member objects but also controls the creation and Deconstruction of member objects. In UML, the combination relationship is represented by a solid line with a solid rhombus head, in which the head points to the whole. (6) Navigation. Navigation describes an object navigating through a chain (an instance of association) to access another object, that is, setting navigation attributes to an associated endpoint means that the object can access the object at the end at the other end. Arrows can be added to the relationship to indicate the direction of navigation. An association that can be navigated only in one direction is called a Unidirection Association, which is represented by a solid line with an arrow.

4. Realization: Realization is the relationship between specification and its implementation, which connects one model element to another, such as classes and interfaces. Generalization and implementation can link a general description with a specific description. Generalization connects elements at the same semantic level and is usually within the same model. Implementing relationships connect elements in different semantic layers, usually based on different models. Realization relationships are usually used in two situations: between interfaces and classes that implement the interface; between use cases and collaboration that implements the use case. In UML, the symbols for realization relationships are similar to those for generalizing relationships, represented by a dashed line with a hollow triangular arrow pointing to the

interface.

2.5.5 Project UML Class Diagram

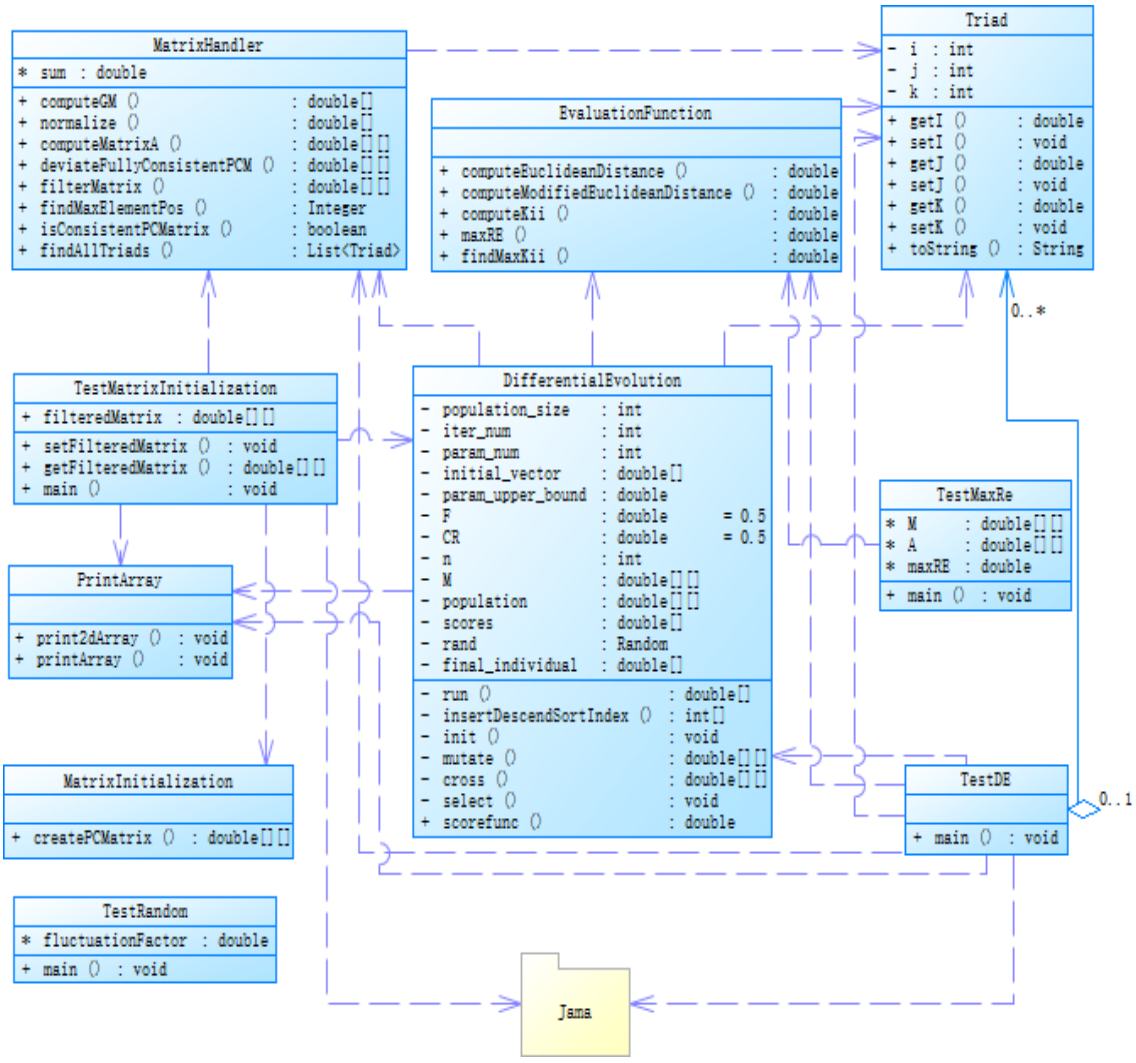


Figure 3: UML Class Diagram

3 Programming Languages Used in The Experiment

3.1 Java

3.1.1 Overview of Java Programming Language

Java is the generic name of Java Object-Oriented Programming Language and Java Platform launched by Sun Microsystems [13]. It was developed by James Gosling and his colleagues and officially launched in 1995 [4]. Java, formerly known as Oak, was designed for embedded chips in consumer electronics in 1991. In 1995, it was renamed Java and redesigned to develop Internet applications. The Hot Java browser implemented in Java (supporting Java applet) shows the charm of Java: cross-platform, dynamic Web, Internet computing. Since then, Java has been widely accepted and promoted the rapid development of the Web. The standard browsers all support Java applet.

On the other hand, Java technology is updated continuously. Java has been very popular and developed fast since it came out, which has a substantial impact on C++ language. In the industrial environment of global cloud computing and mobile internet, Java has more significant advantages and broad prospects.

3.1.2 Characteristics of Java

1. Simplicity

   The syntax of Java language is very close to C language and C++ language, which makes it easy for most programmers to learn and use Java. Java, on the other hand, discards features that are rarely used, difficult to understand, and confusing in C++. Many complex functions in C and C++ are removed, such as a pointer, operator overload, multiple inheritances, automatic forced type conversion, no go statement, no struct and union. In particular, the Java language does not use pointers and provides automatic garbage collection so that programmers do not have to worry about memory management.

2. Object-oriented

   Object-oriented can be said to be the most essential feature of Java. The design of the Java language is entirely object-oriented. It does not support process-oriented programming technology like C language. All Java programs and applets are objects. Java supports static and dynamic code inheritance and reuse. It is more thorough and pure than C++. Java language provides primitives such as class, interface, and inheritance. For simplicity, it only supports single inheritance between classes, but supports multiple inheritances between interfaces, and supports the implementation mechanism between classes and interfaces (keyword is implemented).

3. Cross-platform

   Platform-independent is the most significant advantage of the Java language. A significant problem faced by programs written in other languages is that changes in operating systems, processor upgrades and changes in core system resources can lead to errors or failures in programs. Java Virtual Machine has successfully solved this problem. Programs written in Java can run correctly on any computer with Java Virtual Machine JVM installed. Sun Company has achieved its goal of "write once; run everywhere."

4. Robustness

   Java's robust typing mechanism, exception handling, and automatic collection of waste are essential guarantees for the robustness of Java programs. Discarding pointers is a wise choice for Java. Java's security checking mechanism makes Java more robust.

5. Safety

   As a network language, security is critical. Java security can be guaranteed in two ways. On the one hand, in Java language, C++ functions such as pointer and memory release are deleted to avoid illegal memory operations. On the other hand, when Java is used to create browsers, language functions are combined with functions provided by a class of browsers themselves to make it safer. The Java language has to be tested many times before it can be executed on the machine. It passes

code validation, checks the format of code segments, detects pointer operations, whether object operations are excessive, and attempts to change the type of an object. Besides, Java has many levels of interlocking protection measures, which can effectively prevent the invasion and destruction of viruses.

6. Multithreading

   In the Java language, threads are particular objects that must be created by Thread classes or their descendants. There are usually two ways to create threads: one is to wrap an object that implements the Runnable interface into a thread using a constructor constructed as Thread (Runnable). The other is to derive a subclass from the Thread class and rewrite the run method. The object created with this subclass is a thread. It is noteworthy that the Thread class has implemented the Runnable interface, so any thread has its run method, which contains the code that the thread is running. A set of methods controls thread activity. Java language supports simultaneous execution of multiple threads and provides a synchronization mechanism between threads (keyword is synchronized).

7. Dynamism

   One of the design goals of the Java language is to adapt to the dynamic environment. The classes required by Java programs can be loaded dynamically into the running environment or through the network. This

is also conducive to the software upgrade. Besides, classes in Java have a runtime representation that allows runtime type checking.

3.1.3 Disadvantages

Of course, everything can not be perfect. There are some disadvantages.

1. Efficiency

   Because it has to be compiled and the garbage collection mechanism, so the speed is relatively slow! Not suitable for large-scale programming, online games.

2. Complexity

   Because of its powerful function, it also increases its complexity. There are many popular frameworks, such as struts, spring, jQuery and so on, which undoubtedly increases the complexity of java.

3.1.4 Running Mechanism of Java

1. Operating mechanism of other high-level languages

   Computer high-level language can be divided into compiler type and interpreter type according to the way of program execution.

   Compiled language refers to a separate compilation process before the program is executed. For a specific platform (operating system), the high-level language will be translated into machine language [4, 9].

When the program is executed later, it can run directly, so that the compiled language can run independently from the development environment, and the efficiency can be higher. However, there is also a disadvantage, that is, the compiled language is compiled into machine code on a specific platform, which can not be directly transferred to other platforms to run. If there is a need, it is necessary to re-transfer the source code to a specific platform, after modifying some platforms, to recompile. C/C++ and so on are compiled languages.

Interpretative language refers to the translation of programs into machine language at runtime, which is cross-platform, but the disadvantage is that every executive needs to be compiled once [21, 7]. It is equivalent to mixing the compilation and interpretation processes in a compiled language at the same time. Ruby and Python are interpretative languages.

2. Running mechanism of Java program

   Java language is unique. Programs which are written in Java language also need compiling steps, but it is not compiled into the machine code of a specific language but produces machine-independent bytecode (*. class file). This bytecode cannot run directly and needs to run through the Java interpreter (JVM Java Virtual Machine). Therefore, the Java language compiles and interprets the two steps separately.

3. Java virtual machine

When a Java compiler compiles Java code, it generates bytecode oriented to JVM, and then JVM is oriented to various operating systems. Therefore, programs which are written in Java language run on JVM, not on the operating system. It has an interpreter component that enables communication between Java bytecode and computer operating system.

JVM is similar to an abstract computer. Like a real computer, it has instruction sets and uses different storage areas — complex execution of instructions to manage data, memory, and registers.

3.1.5 Java Development Kit

JDK is the abbreviation of Java development toolkit. It is a set of development kits provided by Sun for developing Java programs. It provides various tools and resources for compiling and running Java programs. It includes Java compiler, Java runtime environment, standard Java class libraries.

3.1.6 Java Runtime Environment

JRE is a prerequisite for running Java programs. Jre contains JVM, which is the core virtual machine for running Java programs. Running Java programs, not only the core virtual machine but also the class loader, bytecode verifier and a large number of necessary class libraries are needed. JDK also contains development tools. So if someone needs to run Java programs, they only need to install Jre.

3.2 R

3.2.1 Overview of R Programming Language

R is the world's most popular language for developing statistical software [30] and R is a branch of S language which was widely used in the field of statistics and was born around 1980. It can be considered that R is an implementation of the S language. S language is an interpretative language developed by AT&T Bell Laboratory for data exploration, statistical analysis, and mapping. The original implementation version of S language was mainly S-PLUS. S-PLUS is commercial software, which is based on S language and further improved by MathSoft's Department of Statistical Science. Later, Robert Gentleman and Ross Ihaka of the University of Auckland in New Zealand and other volunteers developed an R system. R Development Core Team is responsible for the development. R can be seen as an implementation of S language developed by Rick Becker, John Chambers and Allan Wilks of AT&T Bell Laboratories. Of course, S language is also the basis of S-Plus. So, they are almost the same in program grammar, maybe only slightly different in function. Programs can be easily transplanted into a program, and many of them can be applied to R with a little modification [14].

3.2.2 Characteristics of R

1. Free software

   R is a completely free and open source. Everybody can download any

relevant installer, source code, package, source code and documentation from its website and its image. The standard installation file itself has many modules and built-in statistical functions, which can directly implement many commonly used statistical functions after installation.

2. Programmable language

   As an open statistical programming environment, grammar is easy to understand and master. Moreover, after learning, we can compile our functions to extend the existing language. That is why it updates faster than general statistical software, such as SPSS, SAS and so on. Most of the latest statistical methods and techniques can be obtained directly in R.

3. Encapsulation

   All R functions and data sets are stored in the package. Only when a package is loaded can its contents be accessed. Some common and basic packages have been included in the standard installation files. With the emergence of new statistical analysis methods, the packages contained in the standard installation files are continually changing with the updates of versions. In the other version of the installation file, the packages already included include the basic module of base-R, the maximum likelihood estimation module of mle, the ts-time series analysis module, the mva-multivariate statistical analysis module, the survival-life analysis module and so on.

4. Highly interactive

   In addition to graphics, the output is in another window; its input and output windows are all in the same window. If there are errors in the input grammar, it will be prompted in the window immediately. It has a memory function for the previously inputted commands and can be reproduced, edited and modified at any time to meet the needs of users. The output graphics can be directly saved as JPG, BMP, PNG and other image formats, and can also be directly saved as PDF files. Besides, there is an excellent interface with other programming languages and databases.

3.2.3 Language Environment of R

R is a set of data operation, calculation and graphics display functions integrated into a suite.

The purpose of using "environment" here is to show that the positioning of R is a perfect and unified system, not as a specialized and inflexible accessory tool as other data analysis software.

3.2.4 Functions of R

R is a complete software system for data processing, calculation, and mapping. Its functions include: data storage and processing system; array computing tools (especially powerful in vector and matrix operations); complete and coherent statistical analysis tools; excellent statistical mapping func-

tions; simple and powerful programming language: can manipulate the input and output of data, can achieve branch, cycle, user-defined functions.

R is not so much a statistical software as a mathematical computing environment, because R does not only provide many statistical procedures, users only need to specify a database and many parameters to carry out a statistical analysis. The idea of R is that it can provide some integrated statistical tools, but more importantly, it can provide a variety of mathematical calculation, statistical calculation functions, so that users can flexibly and flexibly carry out data analysis, and even create new statistical calculation methods that meet the needs [30].

The grammar of the language is similar to C on the surface, but semantically it is a variant of functional programming language and has strong compatibility with Lisp and APL. In particular, it allows computing on the language, which makes it possible to use expressions as input parameters of functions, which is very useful for statistical simulation and plotting.

R is free software, and it has UNIX, LINUX, MacOS, and Windows versions, which are free to download and use. There are only eight basic modules in R's installation program, and CRAN can obtain other external modules.

R's source code can be downloaded freely, and a compiled version of the executable can be downloaded. It can run on a variety of platforms, including UNIX (including FreeBSD and Linux), Windows and MacOS. R operates mainly on the command line, and several graphical user interfaces have been developed.

R has built-in functions of statistics and digital analysis. Because of S's lineage, R has a stronger object-oriented (object-oriented) function than other statistical or mathematical programming languages.

Another strong point of R is the drawing function, which has the quality of printing and can also add mathematical symbols.

Although R is mainly used for statistical analysis or development of statistical software, it is also used for matrix calculation. Its analysis speed is comparable to that of GNU Octave and even commercial software MATLAB.

User-written suites can enhance r's functionality. The added functions include unique statistical technology, drawing function, programming interface and data output/input function. These packages are written by R, LaTeX, Java and the most commonly used C and Fortran. Many packages with core functions will accompany the downloaded version of the execution file, and according to CRAN records, there are thousands of different packages. Several of them are commonly used, such as econometrics, financial analysis, Human Sciences, and artificial intelligence.

3.2.5 R Package and Its Applications

Many R functions can be used to manage packages. The first time installing a package, use the command "install. packages ()". Providing the package name directly as a parameter to this function. For example, a function for building enhanced scatterplots is provided in package gclus. Programmers can use the command "install. packages ('gclus')" to download and install

it. A package only needs to be installed once. However, like other software, packages are often updated by their authors. The installed packages can be updated with the command "update. packages ()". At the same time, programmers can also click on packages at the bottom right of RStudio, then click install, and enter the package name in the dialog box to download the installation package.

Package installation refers to the process of downloading it from a CRAN mirror site and putting it into the library. To use it in the R session, it is also needed to load the package using the library () command. For example, to use the gclus package, execute the command library (gclus). Of course, a package must have installed before loading it. In a session, the package only needs to be loaded once. If necessary, the boot environment can be customized to load those packages that will be frequently used automatically.

After loading a package, a series of new functions and data sets can be used. Packets often provide demonstrative small data sets and sample code that allow us to try out these new features. The help system contains a description of each function (with examples) and information for each data set. The command help (package= "package_name") can output a short description of a package and a list of function names and dataset names in the package. The function "help ()" can be used to see more details about any function or data set in it. This information can also be downloaded from CRAN in the form of PDF help manuals.

4 IDEs Used in The Experiment

4.1 IntelliJ IDEA

IntelliJ IDEA is an integrated environment for Java programming language development. IntelliJ is recognized as one of the best Java development tools in the industry, especially in intelligent code assistant, code automatic prompt, refactoring, J2EE support, various versions of tools (git, svn...), JUnit, CVS integration, code analysis, innovative GUI design, and other functions can be said to be extraordinary [22].

IDEA is the product of JetBrains, a company headquartered in Prague, the capital of the Czech Republic [10]. Its developers are predominantly Eastern European programmers known for their rigour. Its flagship version also supports HTML, CSS, PHP, MySQL, Python, etc. The free version only supports a few languages, such as Java.

Figure 4: IntelliJ IDEA

4.1.1 Special Functions

1. Rich navigation modes:

   IDEA provides rich navigation modes, such as Ctrl + E displaying recently opened files, Ctrl + N displaying the class name lookup box that programmers want to display (the box also has an intelligent complementary function, IDEA will display all candidate class names when you enter letters). In the most basic project view, programmers can also choose a variety of views.

2. Historical recording function:

   Without using a version management server, a simple IDEA can view the history of files in any project, and programmers can easily restore them when the version is restored.

3. Perfect support for JUnit:

   JUnit is a unit testing framework for Java language. Established by Kent Beck and Erich Gamma, it has gradually become the most successful xUnit family of sUnit originating from Kent Beck. JUnit has its JUnit extended ecosystem. Most Java development environments have integrated JUnit as a unit testing tool

4. Superior support for refactoring:

   IDEA is the first IDE to support refactoring, and its excellent refactoring capability has always been one of its main selling points.

5. Code aided:

   The toString (), hashCode (), equals (), and all get/set methods advocated in the Java specification allow you to automatically generate code without any input, thus freeing you from boring basic method coding.

6. Flexible typesetting function:

   Almost all IDEs have reprogramming functions, but the only IDEA is human, because it supports customization of typesetting mode, programmers can use different typesetting methods according to different project requirements.

7. Perfect support for XML:

   Prompt support for XML: All popular frameworks support full prompt for XML files

8. Dynamic grammar detection:

   Any inconsistent with Java specifications, their predefined specifications, cumbersome will be highlighted in the page.

9. Code checking:

   Automated code analysis, detection of non-conforming, risky code, and highlight.

10. Full support for JSP:

    No plug-ins are required, and JSP is fully supported.

11. Intelligent editing:

    Automatically supplement methods or classes during code input.

12. EJB support:

    No plug-ins are required to support EJB (6.0 supports EJB 3.0) fully

13. Column editing mode:

    UltraEdit certainly appreciates its column editing mode because it reduces much tedious duplication of work, and IDEA fully supports this mode, thus improving coding efficiency.

14. Preset template:

    Prefabricated templates allow programmers to edit frequently used methods into templates. When programmers use them, programmers can complete the entire code by typing only a few letters. For example, using a higher public static void main (String [] args) you can preset PM as the method in the template. When programmers enter pm, programmers press the code assist key. IDEA will complete the automatic input of the code.

15. Perfect automatic code completion:

    A quick inspection of methods in classes automatically completes code input when only one method name is found, thus reducing the writing of the remaining code.

16. Version control perfect support:

It integrates all the common plug-ins of version control tools, including git, SVN, and GitHub, so that developers can submit, check out, resolve conflicts, view the content of version control server directly in IntelliJ idea in programming engineering.

17. Check for unused code:

    Automatically check code that is not used in the code and give tips to make the code more efficient.

18. Intelligent code:

    Automatically check the code, find out the code that is different from the present specification and gives a prompt. If the programmer agrees to modify, automatically complete the modification. For example, code: String STR = Hello IntelliJ + IDEA; IDEA will give optimization tips, if the programmer agrees to modify IDEA, the code will automatically be changed to String STR = Hello IntelliJ IDEA;

19. Finding and replacing functions of regular expressions:

    Finding and replacing support regular expressions to improve efficiency.

20. JavaDoc preview support:

    Support JavaDoc preview function, Ctrl + Q display JavaDoc results in JavaDoc code, thereby improving the quality of doc documents.

21. Programmers intend to support:

    When programmers code, IDEA detects programmers' intentions, pro-

vides suggestions, or directly helps programmers to complete the code.

4.1.2 Advantages

The most prominent function is debugging (Debug), which can debug Java code, JavaScript, JQuery, Ajax and other technologies. For example, if a programmer looks at an object of Map type, if the implementation class uses hash mapping, it automatically filters empty Entry instances.

Secondly, we need to evaluate the value of an expression dynamically. For example, a programmer gets an instance of a class, but the programmer does not know its API. We can point out the methods it supports through Code Completion.

Finally, in the case of multithreaded debugging, the Logon console function can help programmer check the execution of multithreaded.

4.1.3 Versions

IntelliJ IDEA 1.0 was released in January 2001 and 2.0 in July of the same year, followed by an essential annual release (except 2003), of course, each version is upgraded every year. After version 3.0, IDEA has won many awards, among which "Jolt Productivity Award" and "Java World Editors'Choice Award" won in 2003 are the symbols, thus establishing the status of IDEA in IDE.

IntelliJ IDEA is divided into Ultimate Edition flagship version and Community Edition version. Flagship version can be tried for 30 days for free.

The community version is free to use, but the function is reduced compared with the Flagship version.

4.1.4 System Requirements

- Microsoft Windows 8/7/Vista/2003/XP (incl.64-bit)
- 1 GB RAM minimum, 2 GB RAM recommended
- 300 MB hard disk space + at least 1 G for caches
- 1024*768 minimum screen resolution
- JDK 1.6 or higher

4.2 RStudio

RStudio is a new Integrated Development Environment (IDE) for the R programming language [1]. Compared with RGui, Rstudio is much more convenient to use as an editor of the R language. Rstudio is a free open source r integrated development environment, a programming language for statistical computing and graphics. RStudio also allows programmers to execute code directly in the resource editor. For most R developers, this is the best way to reproduce code. People can copy much code in the editor or reuse package commands as reuse functions.

Figure 5: RStudio

4.2.1 Management Documents

- R script
- R Markdown file
- Sweave document
- HTML file
- TeX document

4.2.2 Characteristics of RStudio

- RStudio supports tab keys to complete the code automatically. For example, if programmers want to enter poll into their workspace, they can enter poll, and then Tab, RStudio will automatically complete all the code. In the console code, programmers can also use this way to complete the code automatically [38].
- RStudio supports to find and replace functions in the resource editor:

- RStudio automatically parses the code in the resource editor, identifies it and converts it into a reusable function, where variables automatically become parameters of the function.

- When editing reusable functions, you want to save resources like documents, and you have the save key Source on Save in the toolbar. When selected, all resources will be saved to the global environment and saved at any time.

4.2.3 System Requirements

- Microsoft Windows 8/7/Vista/2003/XP (incl.64-bit)
- 1 GB RAM minimum, 2 GB RAM recommended
- 300 MB hard disk space + at least 1 G for caches
- 1024*768 minimum screen resolution
- JDK 1.6 or higher

5 Pairwise Comparisons Matrices

5.1 Introduction of Pairwise Comparisons

Pairwise comparisons (PC) generally is any process of comparing entities in pairs to judge which of each entity is preferred or has a higher amount of quantitative property, or whether or not the two entities are identical.

Pairwise comparisons allow one to represent such subjective assessments and to process them by analyzing and quantifying [16].

The method of pairwise comparisons is often used in the scientific study of preferences, attitudes, voting systems, social choice, public choice, requirements engineering, and multi-agent AI systems. In the psychology literature, it is often referred to as paired comparison. We can use $weight_i$ to represent how much someone likes something or how important something is to someone. Similarly, $weight_j$ to represent how much someone likes the other thing or how vital the other thing is to someone.

The ratio $weight_i/weight_j$ can be a little bigger than one, which means the subject like object i a little more than object j or object i is slightly more important than object j for the subject.

The ratio $weight_i/weight_j$ can also be far higher than one, which means the subject like object i much more than object j or object i is much more important than object j for the subject.

Besides, the ratio in the PC matrix is transitive. For example, if the value of A/B is two and the value of B/C is three, then the value of A/C is six. Reflected in the element position of PC matrix is: if $a_{ij} = 2$ and $a_{jk} = 3$, then $a_{ik} = 6$. If a PC matrix fully satisfies the above rules, we call it fully consistent PC matrix, otherwise, we call it inconsistent PC matrix. The Koczodaj Inconsistency Indicator (Kii) can be used to measure the consistency of a given PC matrix, which will be described in detail in the following chapters.

The ratio $weight_i/weight_j$ always equals one when we compare the same

objects whoever is concerned. For the case of ratio less than one, contrary to the case of a ratio greater than one, and this case will not be repeated here. PC allows us to express preferences more efficiently and more accurately [9].

5.2 Definition of PCM

When determining the weights of factors at different levels, if only qualitative results, it is often not easy to be accepted by others, so Santy et al. put forward a consistent matrix method. Namely: 1. Do not put all the factors together, but compare between two; 2. The relative scale should be adopted to reduce the difficulty of comparing various factors of different nature to improve accuracy.

Defining an n∗n pairwise comparisons matrix (PCM) simply as s square matrix $A=|a_{ij}|$ such that $a_{ij}>0$ for every $i = 1, 2, \cdots, n$. A pairwise comparisons matrix A is called reciprocal if $a_{ij} = 1/a_{ij}$ for every $i = 1, 2, \cdots, n$. [7] (then automatically $a_{ii} = 1$ for every $i = 1, 2, \cdots, n$.). Let

$$\begin{pmatrix} 1 & a_{12} & \dots & a_{1n} \\ 1/a_{12} & 1 & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ 1/a_{1n} & 1/a_{2n} & \dots & 1 \end{pmatrix}_{n\times n}$$

where a_{ij} expresses an expert's relative preference of stimuli s_i over s_j.

It is worth noting that the element value on the principal diagonal of the PC matrix always equals to one because it represents two identical entities for pairwise comparison.

5.3 Consistent PCM and Inconsistent PCM

A PC matrix A=$[a_{ij}]$ is called consistent or transitive if for every $(i, j, k) = 1, 2, 3, ..., n$, there is always $a_{ik} * a_{kj} = a_{ij}$. We will refer to equation $a_{ik} * a_{kj} = a_{ij}$ as a "consistency condition". While every consistent PC matrix is reciprocal, the converse is false in general. If the consistency condition does not hold, the PC matrix is inconsistent (or intransitive) [27].

On the contrary, a PC matrix is called inconsistent PC matrix if it fail to satisfy the "consistency condition". As for inconsistent PC matrix, there is way to calculate the degree of the inconsistency, which is Koczkodaj Inconsistency Indicator (Kii): $Kii(x, y, z) = 1 - min(x * z/y, y/(x * z))$ where (x, y, z) is a triad for the given PC matrix. The greater the value of Kii, the higher the degree of inconsistency of PC matrix. As for Kii, we will discuss it in more detail in subsequent chapters.

Inconsistency in pairwise comparisons occurs due to superfluous input data. Only $n - 1$ pairwise comparisons are really needed to create the entire PC matrix for n entities, while the upper triangle has $n(n-1)/2$ comparisons [27]. Inconsistencies are is not necessarily "wrong" as they can be used to improve the data acquisition. However, there is a real necessity to have a

"measure" for it [27].

For practical application, only PC matrices with dimension 3 to 8 are discussed in the experiment.

5.4 Input Processing for PCM

The initial matrix created should be processed before inputting into the differential evolution (DE) system. However, we should decide which kind of method can be used to get the system input, which is a n dimensional vector. After reading relative documents [18, 28, 24] and materials on-line [27, 25, 12], geometric mean of the initial PC matrix and principal eigenvector of the initial PC matrix seems good. Besides, principle eigenvector of the initial PC matrix is better than the geometric mean of the initial PC matrix.

It is worth noting that when the number of rows or columns of the PC matrix equals three, the geometric mean of the PC matrix is precisely the same as its principal eigenvector. That is to say, when $x = 3$, using the geometric mean of PC matrix and principal eigenvector of the PC matrix as input values of the DE system, we will get the same result. When the number of rows or columns of the PC matrix is greater than three, the situation will be different. The following experimental codes further validate this conclusion.

5.5 Geometric Mean (GM)

The geometric mean is the root of several times of the continuous production of the values of each variable. The method of calculating geometric mean is called the geometric mean method [6]. If the total level and the total result are equal to the sum of all stages, links and successive products, the general level and results of each stage and link should be calculated by the average geometric method instead of a standard arithmetic method. According to the different forms of data, it can be divided into simple geometric mean and weighted geometric mean.

$$GM = \begin{bmatrix} W_0 \\ W_1 \\ W_2 \\ W_3 \\ \vdots \\ W_n \end{bmatrix} \tag{7}$$

$$W_i = (a_1 * a_2 * ... * 1 * ... * a_n)^{1/n} \tag{8}$$

The geometric mean (GM) input vector is an $n * 1$ matrix and W_i is the geometric mean for each row of the original matrix. In the project, GM after

normalization is the input of the differential evolution system.

Computing geometric means for each row in the PC matrix. Two-dimensional array in java was used to represent the PC matrix to be calculated. The return of "computeGM()" function is the geometric means for each row in the input PC matrix.

```java
/**
 * Computes geometric means for each rows in the matrix.
 *
 * @param arr two-dimensional array to be calculated
 * @return geometric means for each rows in the matrix
 */
public static double[] computeGM(double[][] arr) {
    double mul = 1.0F;
    // use array to store geometric means each rows.
    double[] eachRow = new double[arr.length];
    System.out.println("GMs before normalization: ");

    for (int i = 0; i < arr.length; i++) {
        for (int j = 0; j < arr[i].length; j++) {
            mul = mul * arr[i][j];
        }
        eachRow[i] = (double) Math.pow(mul, (1.0 / (double) (arr[i].length)));
        System.out.println(eachRow[i]);
        mul = 1.0F;
        sum += eachRow[i];
    }
    return eachRow;
}
```

Figure 6: computeGM() Function

Normalizing geometric means for each row in the PC matrix and using normalized geometric means as input to the differential evolution algorithm.

As what is shown in Figure 7, the input of the function is the vector need to be normalized, and the sum of all elements equals to one after normalization

operation.

The API class BigDecimal, provided by Java in "java.math" package, is used to perform precise operations on numbers with more than 16 significant bits. Dual-precision floating-point variable double can handle 16-bit valid numbers. In practical applications, it is necessary to calculate and process larger or smaller numbers.

Float and double can only be used for scientific calculation or engineering calculation. In economic calculation, java.math.BigDecimal is used. BigDecimal creates objects. We cannot use traditional arithmetic operators such as $+, -, *, /$ to directly perform mathematical operations on them. Instead, we must call their corresponding methods. The parameters in the method must also be BigDecimal objects. Constructors are unique methods of classes that are used to create objects, especially those with parameters.

There are lots of different constructors in BigDecimal class, but this one: "public BigDecimal(String value)" is recommended to use according to Java API because other constructors may lead to the unpredictability of the results.

```java
/**
 * Normalizes geometric means for each rows in the matrix.
 *
 * @param vec vector need to be normalized
 * @return vector after normalization
 */
public static double[] normalize(double[] vec) {
    double[] normalizedVec = new double[vec.length];
    System.out.print("GMs after normalization: ");

    /*
     * There are lots of different constructors in BigDecimal class, but this one:
     * "public BigDecimal(String val)" is recommended to use according to Java API.
     * Because other constructors may lead to the unpredictability of the results.
     * */
    BigDecimal accurateSum = new BigDecimal(String.valueOf(sum));

    for (int i = 0; i < vec.length; i++) {
        // BigDecimal is used to improve the accuracy in calculation.
        BigDecimal accurateEach = new BigDecimal(vec[i]);
        BigDecimal eachAfterNormalize = accurateEach.divide(accurateSum, scale: 5, BigDecimal.ROUND_HALF_UP);
        normalizedVec[i] = eachAfterNormalize.doubleValue();
        if (i == 0) {
            System.out.print("(" + eachAfterNormalize + ", ");
        } else if (i == vec.length - 1) {
            System.out.println(eachAfterNormalize + ")");
        } else {
            System.out.print(eachAfterNormalize + ", ");
        }
    }
    return normalizedVec;
}
```

Figure 7: normalize() Function

5.6 Principal Eigenvector (EV)

5.6.1 Eigenvalues and Eigenvectors

In linear algebra, an eigenvector vector of a linear transformation is a non-zero vector that changes by only a scalar factor when that linear transformation is applied to it. More formally, if T is a linear transformation from

a vector space V over a field F into itself and v is a vector in V that is not the zero vector, then v is an eigenvector of T if $T(v)$ is a scalar multiple of v. This condition can be written as the equation:

$$T(v) = \lambda v \tag{9}$$

λ is a scalar in the field F, known as the eigenvalue, characteristic value, or characteristic root associated with the eigenvector v.

If the vector space V is finite-dimensional, then the linear transformation T can be represented as a square matrix A, and the vector v by a column vector, rendering the above mapping as a matrix multiplication on the left-hand side and a scaling of the column vector on the right-hand side in the equation [5]:

$$Av = \lambda v \tag{10}$$

There is a direct correspondence between n-by-n square matrices and linear transformations from an n-dimensional vector space to itself, given any basis of the vector space. For this reason, it is equivalent to define eigenvalues and eigenvectors using either the language of matrices or the language of linear transformations.

5.6.2 Eigenvector and Principal Eigenvector

A matrix can have multiple eigenvalues. Among these eigenvalues, the eigenvalue with the largest modulus is the principal eigenvalue (for real matrix, the eigenvector with the largest absolute value), and the eigenvector corresponding to the principal eigenvalue is called the principal eigenvector.

5.6.3 Jama Package

JAMA is a basic linear algebraic java package that provides real non-sparse matrix classes that programmers can construct and manipulate. For coders who often use matrix operations, it does not matter if they are not proficient in linear algebra, because the functions provided by JAMA package are enough, convenient to call, natural to use and easy to understand. The Jama package is intended to be called the Standard Matrix Pack for Java, which is scheduled to be submitted to the Java Grande Forum and then to Sun. Potential competitors for the Java matrix class include the matrix class implemented by Mathworks and the National Standardization Management Committee (NIST). We have released this version for a comprehensive review. Future versions of JAMA may not be compatible with current versions [19].

NIST and the University of Maryland have developed a similar matrix package: Jampack. These two packages emerge because of the different implementation needs of users. Jama is a strict object-oriented framework based on a single matrix class, while Jampack's scheme is more open and

user-friendly. For general users, the two packages differ only at the grammatical level of matrix operations.

Jama consists of six classes: Matrix, Cholesky Decomposition, LU Decomposition, QR Decomposition, Singular Value Decomposition and Eigenvalue Decomposition.

Matrix class provides the essential function of the linear algebraic numerical operation. Different constructors can construct two-dimensional arrays with double precision and floating-point precision, while different gets and

The set method can return the sub-matrix and matrix elements. The basic arithmetic operations include matrix addition, matrix multiplication, matrix norm and arithmetic operations based on matrix elements. Print matrix functions are also included.

Five decompositions of matrices, including one pair or three tuples, permutation vector matrices and so on, correspond to five matrix decomposition classes of JAMA. These decomposition classes can be accessed by Matrix class. They can solve linear equations, matrix determinants, Inverses, and other matrix operations.

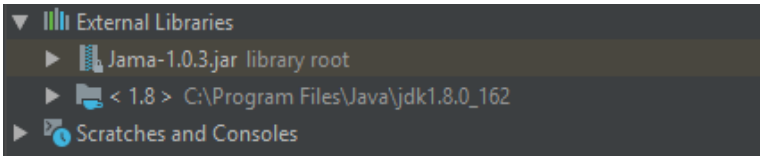


Figure 8: Jama Package

It is convenient to use JAMA to find both Eigen Value and Eigen Vector

of the PCM. JAMA is a basic linear algebraic java package that provides real non-sparse matrix classes that programmers can construct and manipulate. For coders who often use matrix operations, it does not matter if they are not proficient in linear algebra, because the functions provided by JAMA package are enough, convenient to call, natural to use and easy to understand. The Jama package is intended to be called the Standard Matrix Pack for Java, which is scheduled to be submitted to the Java Grande Forum and then to Sun. Potential competitors for the Java matrix class include the matrix class implemented by Mathworks and the National Standardization Management Committee (NIST). We have released this version for extensive review. In the future, the new version of JAMA is not necessarily compatible with the current version.

```
41          Matrix A = new Matrix(M);
42
43          // find the principle eigen vector for input matrix M
44          // A.eig().getD().print(3, 3);
45          // A.eig().getV().print(3, 3);
46          double[][] eigMatrix = A.eig().getV().getArray();
47
48          // TODO
49          // double sumEig = eigMatrix[0][0] + eigMatrix[1][0] + eigMatrix[2][0]+ eigMatrix[3][0];
50          // double[] Matrix2 = {eigMatrix[0][0]/sumEig, eigMatrix[1][0]/sumEig, eigMatrix[2][0]/sumEig, eigMatri
51          int maxElementColumn = MatrixHandler.findMaxElementPos(A.eig().getD().getArray());
52          double sumEig = eigMatrix[0][maxElementColumn]
53                  + eigMatrix[1][maxElementColumn]
54                  + eigMatrix[2][maxElementColumn];
55          double[] Matrix2 = {eigMatrix[0][maxElementColumn] / sumEig,
56                  eigMatrix[1][maxElementColumn] / sumEig,
57                  eigMatrix[2][maxElementColumn] / sumEig};
58
59          for (int i = 0; i < Matrix2.length; i++) {
60              Matrix2[i] = (double) Math.round(Matrix2[i] * 100000) / 100000;
61          }
62          System.out.print("principal EV for matrix M: ");
63          System.out.print("(");
64          PrintArray.printArray(Matrix2);
65          System.out.println();
66
67          double[] initial_vector = Matrix2;
```

Figure 9: Use Jama Package to Find Principal Eigenvector in PCM

6 The Parameters in DE

Research on parameter influence of differential evolution algorithm is essential. The performance of the differential evolution algorithm is mostly related to the selection of parameters. However, there are a few particular articles on the parameter analysis of the DE algorithm. In this paper, three main parameters affecting the performance of the DE algorithm are studied and analyzed, and some reasonable selection rules are given.

1. Population Size (NP)

   From the computational complexity analysis, the larger the population size, the higher the possibility of searching for the optimal global solution. However, the computational complexity and computational time required will also increase. Moreover, the quality of the optimal solution does not blindly improve with the increase in population size. Sometimes, the increase in population size will reduce the accuracy of the optimal solution. Therefore, a reasonable selection of population size is of considerable significance to improve the search efficiency of the algorithm.

   When the population size increases to a certain number, the accuracy of the solution will not improve, or even decrease. Because a larger population size can maintain the diversity of the population but will reduce the convergence rate, diversity and convergence speed must maintain a correct balance. Therefore, if the population size is too large, the

accuracy will decrease if the maximum evolution algebra is not added. Besides, the larger the population size, the higher the diversity, so if the population converges prematurely, it is necessary to increase the population size to increase diversity.

Given the maximum evolutionary algebra, the population size is better between [15, 35] for low-dimensional simple problems and [30, 50] for high-dimensional complex problems. In short, given the maximum evolutionary algebra, when the population size is between [15, 50], it can maintain the appropriate balance between diversity and convergence rate.

2. Mutation Factor (F)

   When the scaling factor F is between [0.5, 1], the result obtained by the algorithm is better; when $F \leq 0.5$ or $F \geq 1$, the quality of the solution obtained by the algorithm is not high.

   The scaling factor F is used to control the influence of the differential vector on the V of the mutant. When F is large, the difference vectors have a more significant impact on V and can produce more massive disturbances, which is conducive to maintaining the diversity of the population. On the contrary, when F is small, the disturbance is small, and the scaling factor can play the role of the local search. Therefore, F played a specific role in regulating population diversity. The scaling factor F is too large to maintain the diversity of the population. The

algorithm is approximate to random search, the search efficiency is low, and the accuracy of the optimal global solution is low; conversely, F is too small, the diversity of the population will lost very quickly, and the algorithm is easy to trap the local optimum and appear premature convergence [37].

Because the DE algorithm is a "greedy" selection algorithm, with the continuous evolution of the population, all the individuals gradually approach the optimal individual, and the differences between individuals will gradually decrease. The diversity of the population will be lost when the algorithm evolves to a certain extent. Population diversity has a positive impact on the global search ability of the algorithm. Population diversity increases the possibility of escaping from the local optimum, which is conducive to global search, but reduces the convergence rate; Population diversity is small, which is conducive to local search, and convergence speed is fast, but it is easy to fall into the local optimum, so-called premature phenomenon occurs.

In summary, F plays a specific role in the local search and global search of the algorithm. The more significant F is beneficial to maintain population diversity and global search, while the smaller F is beneficial to local search and improve convergence speed.

3. Crossover Rate (CR)

   When the value of CR is small, the number of evaluations of function

needed is larger, and the convergence speed is slower, but the success rate is higher, and the stability of the algorithm is better. When the value of CR is large, the convergence is often accelerated, but it is easy to fall into local optimum, and premature phenomena occur. The success rate of achieving the given accuracy is low, and the stability is poor. It can be seen that the success rate and the speed of convergence are a pair of contradictions. Therefore, in order to ensure higher success rate and faster convergence rate, for single-peak functions, CR values should be between [0.6, 0.8], and for other complex multi-peak functions, CR values should be between [0.1, 0.5].

The U of the new generation is produced by the intersecting of the V between the mutant individuals and the X components of the parent individuals. The larger the value of CR, the more contribution V makes to U pairing, which is conducive to opening up new space and accelerating convergence, but in the later stage, the variant individuals tend to converge. It is not conducive to maintaining diversity, so it is accessible to prematurity and poor stability. The smaller the CR value, the more contribution X makes to U, which weakens the ability of the algorithm to open up new space. The convergence speed is relatively slow, but it is conducive to maintaining the diversity of the population so that it can have a higher success rate.

7 Experiment Notes

7.1 Euclidean Distance

In mathematics, Euclidean distance or Euclidean measure is the "ordinary" (straight line) distance between two points in Euclidean space. Euclidean space becomes metric space by using this distance. The associated norm is called the Euclidean norm. Earlier literature called Pythagorean Metrics.

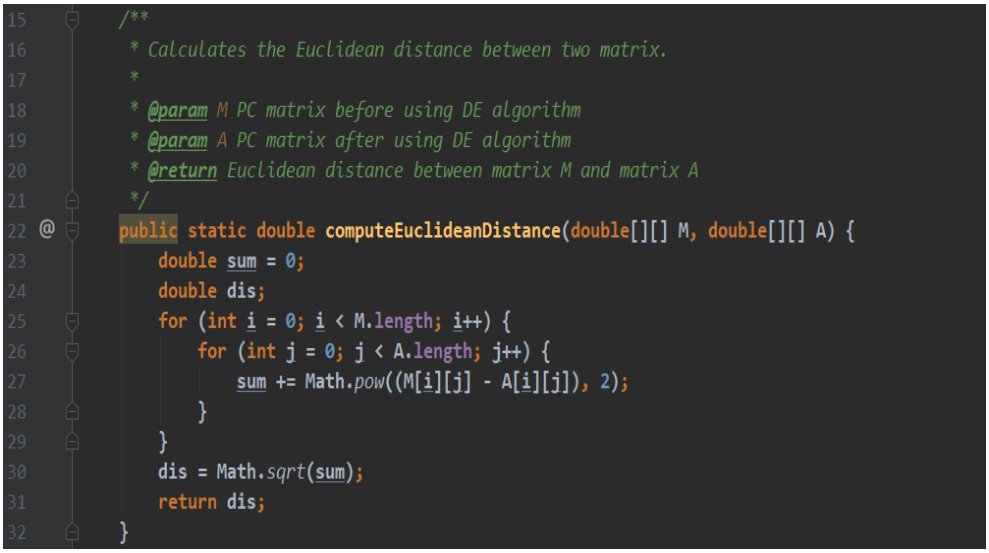
The figure 10 shows the Euclidean formula in n-dimensional space:

$$d(x,y) := \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_n - y_n)^2} = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}.$$

Figure 10: Euclidean Formula in N-dimensional Space

The Euclidean distance function is chosen as the evaluation function of the differential evolution system. After each iteration, the new GM vector is obtained, and then the calculation of the Euclidean distance between matrix M (initial input matrix) and matrix A (newly created matrix in generation i) is processed.

As the population continues to iterate, individuals in the population will evolve, which means that the distance between matrices M and A is narrowing [26]. Reasonable results shown below can be obtained the moment we run the code block "testDE.java":



```java
/**
 * Calculates the Euclidean distance between two matrix.
 *
 * @param M PC matrix before using DE algorithm
 * @param A PC matrix after using DE algorithm
 * @return Euclidean distance between matrix M and matrix A
 */
public static double computeEuclideanDistance(double[][] M, double[][] A) {
    double sum = 0;
    double dis;
    for (int i = 0; i < M.length; i++) {
        for (int j = 0; j < A.length; j++) {
            sum += Math.pow((M[i][j] - A[i][j]), 2);
        }
    }
    dis = Math.sqrt(sum);
    return dis;
}
```

Figure 11: computeEuclideanDistance() Function

The test results for case 1:

Matrix M:

1.00000, 2.00000, 5.00000

0.50000, 1.00000, 3.00000

0.20000, 0.33333, 1.00000

The initial vector is: (0.58155, 0.30900, 0.10945)

01(0.49251, 0.39929, 0.10820); ED: 1.168701748623992;

02(0.49251, 0.39929, 0.10820); ED: 1.168701748623992;

03(0.49251, 0.39929, 0.10820); ED: 1.168701748623992;

04(0.49251, 0.39929, 0.10820); ED: 1.168701748623992;

05(0.51831, 0.37888, 0.10282); ED: 0.9629842313857246;

06(0.51831, 0.37888, 0.10282); ED: 0.9629842313857246;

07(0.58559, 0.29824, 0.11617); ED: 0.4398012821234418;

08(0.58559, 0.29824, 0.11617); ED: 0.4398012821234418;

09(0.58559, 0.29824, 0.11617); ED: 0.4398012821234418;

10(0.58559, 0.29824, 0.11617); ED: 0.4398012821234418;

11(0.56202, 0.33022, 0.10776); ED: 0.3838134425435809;

12(0.56202, 0.33022, 0.10776); ED: 0.3838134425435809;

13(0.55520, 0.33542, 0.10938); ED: 0.37408535485946354;

14(0.56456, 0.32269, 0.11275); ED: 0.29526414406158813;

15(0.56929, 0.31773, 0.11299); ED: 0.2899206969602925;

16(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

17(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

18(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

19(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

20(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

21(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

22(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

23(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

24(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

25(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

26(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

27(0.57043, 0.31773, 0.11183); ED: 0.28442420933854506;

28(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

29(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

30(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

31(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

32(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

33(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

34(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

35(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

36(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

37(0.56960, 0.31886, 0.11155); ED: 0.2843099585285853;

38(0.56888, 0.31890, 0.11222); ED: 0.28386947884632263;

39(0.56888, 0.31890, 0.11222); ED: 0.28386947884632263;

40(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

41(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

42(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

43(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

44(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

45(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

46(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

47(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

48(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

49(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

50(0.56880, 0.31918, 0.11203); ED: 0.2833713654297522;

...

490(0.56880, 0.31928, 0.11192); ED: 0.28327461146084937;

491(0.56880, 0.31928, 0.11192); ED: 0.28327461146084937;

492(0.56880, 0.31928, 0.11192); ED: 0.28327461146084937;

493(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

494(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

495(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

496(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

497(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

498(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

499(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

500(0.56880, 0.31931, 0.11189); ED: 0.28327177541097714;

Result:

1. initial M: (0.58155, 0.30900, 0.10945)
2. evolved M: (0.56880, 0.31931, 0.11189)

7.2 Modified Euclidean Distance

Although Euclidean distance is instrumental, it has distinct disadvantages. It equates the differences between different attributes (i.e., indices or dimensions of variables) of samples, which sometimes fails to meet the actual requirements. For example, in educational research, we often encounter the analysis and discrimination of human beings. Different attributes of individuals have different importance in distinguishing individuals. Therefore,

Euclidean distance is suitable for the case of uniform measurement criteria for each component of a vector.

Necessary adjustments to Euclidean distances: there are many weighted Euclidean distances, some regards reciprocal of variance as a weight, and some regards distance as a weighting factor.

Improved Euclidean distance is divided by the square of matrix dimension on the original basis. The result is very similar to the old experiment, but when the number of the dimension of PC matrices increases, the difference of the result increases [25].

```
34    /**
35     * Calculates the modified Euclidean distance between two matrix.
36     *
37     * @param M PC matrix before using DE algorithm
38     * @param A PC matrix after using DE algorithm
39     * @return Modified Euclidean distance between matrix M and matrix A
40     */
41    public static double computeModifiedEuclideanDistance(double[][] M, double[][] A) {
42        double modifiedDis;
43        double dis = EvaluationFunction.computeEuclideanDistance(M, A);
44        modifiedDis = dis / (Math.pow(M.length, 2));
45        return modifiedDis;
46    }
```

Figure 12: computeModifiedEuclideanDistance() Function

The test results for case 2:

Matrix M:

1.00000, 2.00000, 5.00000

0.50000, 1.00000, 3.00000

0.20000, 0.33333, 1.00000

The initial vector(EV/GM) is: (0.58155, 0.30900, 0.10945)

01(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

02(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

03(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

04(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

05(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

06(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

07(0.58257, 0.29717, 0.12026); MED: 0.06196200146698002;

08(0.57405, 0.30783, 0.11812); MED: 0.04929298888057073;

09(0.57405, 0.30783, 0.11812); MED: 0.04929298888057073;

10(0.57405, 0.30783, 0.11812); MED: 0.04929298888057073;

11(0.54948, 0.33722, 0.11330); MED: 0.046269753960815396;

12(0.54948, 0.33722, 0.11330); MED: 0.046269753960815396;

13(0.54948, 0.33722, 0.11330); MED: 0.046269753960815396;

14(0.54948, 0.33722, 0.11330); MED: 0.046269753960815396;

15(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

16(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

17(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

18(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

19(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

20(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

21(0.57349, 0.31044, 0.11607); MED: 0.04097354174366191;

22(0.57763, 0.31310, 0.10927); MED: 0.03949319307426245;

23(0.56977, 0.32046, 0.10977); MED: 0.03446664161455345;

24(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

25(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

26(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

28(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

29(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

30(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

31(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

32(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

33(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

34(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

35(0.56522, 0.32305, 0.11173); MED: 0.0320542604956668;

36(0.56823, 0.32011, 0.11166); MED: 0.031522535637708564;

37(0.56823, 0.32011, 0.11166); MED: 0.031522535637708564;

38(0.56823, 0.32011, 0.11166); MED: 0.031522535637708564;

39(0.56823, 0.32011, 0.11166); MED: 0.031522535637708564;

...

490(0.56880, 0.31931, 0.11190); MED: 0.03147464004691641;

491(0.56880, 0.31931, 0.11190); MED: 0.03147464004691641;

492(0.56880, 0.31931, 0.11190); MED: 0.03147464004691641;

493(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

494(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

495(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

496(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

497(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

498(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

499(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

500(0.56880, 0.31930, 0.11190); MED: 0.03147463983480267;

Result:

1. initial M: (0.58155, 0.30900, 0.10945)
2. evolved M: (0.56880, 0.31930, 0.11190)

7.3 Maximum of Relative Error

The relative error refers to the ratio of the absolute error caused by measurement and divided by the real value of the measured value (multiplied by 100% and expressed in percentage). Generally speaking, the relative error can better reflect the credibility of measurement. Let the measured result y be subtracted from the agreed truth value t, and the error or absolute error is δ. The relative error can be obtained by dividing the absolute error δ by the agreed truth value t.

$$\delta = \Delta / L \times 100\%$$

Figure 13: Relative Error Formula

In case3, maxRE function is chosen as the way to evaluate the individuals each generation[24, 18]:

The algorithm to calculate maxRE between two PCM: Given inconsistent PCM 3 by 3, find an consistence PC matrix A = $[w_i/w_j]$ for which maxRE has the minimum value.

1. The initial value of vector w is GM of rows of PC matrix M;

2. Function maxRE is defined as the maximum value of: $|(m_{ij} - a_{ij})/m_{ij}|$ where $|*|$ is the absolute value and vector w change is not more than c percent (initially, c is plus/minus 10%)

3. Use DEoptim to minimize the function maxRE;

4. Print maxRE, Kii, v_1, v_2, v_3.

```java
/**
 * Calculates the maxRE between matrix M and matrix A.
 *
 * @param M PC matrix before using DE algorithm
 * @param A PC matrix after using DE algorithm
 * @return the value of max relative error between matrix M and matrix A
 */
public static double maxRE(double[][] M, double[][] A) {
    double maxRE = 0;
    for (int i = 0; i < M.length; i++) {
        for (int j = 0; j < M[i].length; j++) {
            double temp = Math.abs((A[i][j] - M[i][j]) / M[i][j]);
            if (maxRE < temp) {
                maxRE = temp;
            }
        }
    }
    // maxRE = (double) Math.round(maxRE * 100000) / 100000;
    return maxRE;
}
```

Figure 14: maxRE() Function

The test results for case 3:

1.00000, 2.00000, 5.00000

0.50000, 1.00000, 3.00000

0.20000, 0.33333, 1.00000

The initial vector(EV/GM) is: (0.58155, 0.30900, 0.10945)

01(0.58746, 0.26329, 0.14925); maxRE: 0.7005600382949742;

02(0.58746, 0.26329, 0.14925); maxRE: 0.7005600382949742;

03(0.56246, 0.36299, 0.07455); maxRE: 0.6230328143620841;

04(0.61150, 0.27503, 0.11346); maxRE: 0.23764163703989308;

05(0.61150, 0.27503, 0.11346); maxRE: 0.23764163703989308;

06(0.61150, 0.27503, 0.11346); maxRE: 0.23764163703989308;

07(0.57148, 0.30754, 0.12098); maxRE: 0.18012343292641914;

08(0.57148, 0.30754, 0.12098); maxRE: 0.18012343292641914;

09(0.61082, 0.28067, 0.10850); maxRE: 0.15974911380901952;

10(0.61082, 0.28067, 0.10850); maxRE: 0.15974911380901952;

11(0.59775, 0.29297, 0.10929); maxRE: 0.11911808882248655;

12(0.59775, 0.29297, 0.10929); maxRE: 0.11911808882248655;

13(0.59775, 0.29297, 0.10929); maxRE: 0.11911808882248655;

14(0.58970, 0.30251, 0.10779); maxRE: 0.09419923253817775;

15(0.58970, 0.30251, 0.10779); maxRE: 0.09419923253817775;

16(0.58970, 0.30251, 0.10779); maxRE: 0.09419923253817775;

17(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

18(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

19(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

20(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

21(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

22(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

23(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

24(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

25(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

26(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

27(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

28(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

29(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

30(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

31(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

32(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

33(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

34(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

35(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

36(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

37(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

38(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

39(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

40(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

41(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

42(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

43(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

44(0.58238, 0.30843, 0.10918); maxRE: 0.06678229795869406;

45(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

46(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

47(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

48(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

49(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

50(0.58204, 0.30856, 0.10940); maxRE: 0.06410463241330788;

...

490(0.58154, 0.30899, 0.10947); maxRE:0.0628851092744405;

491(0.58155, 0.30899, 0.10946); maxRE:0.06275394186298897;

492(0.58155, 0.30899, 0.10946); maxRE:0.06275394186298897;

493(0.58157, 0.30898, 0.10945); maxRE:0.0627458042489641;

494(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

495(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

496(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

497(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

498(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

499(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

500(0.58156, 0.30899, 0.10945); maxRE:0.0626966210015997;

Result:

1. initial M: (0.58155, 0.30900, 0.10945)

2. evolved M: (0.58156, 0.30899, 0.10945)

7.4 Triad and Koczkodaj Inconsistency Indicator (Kii)

It is needed to change using a primitive Euclidean distance or modified Euclidean distance as evaluation function to something a bit making more sense: replacing the fitness function from Euclidean Distance (ED) to Koczkodaj Inconsistency Indicator (Kii), Kii is a method to evaluate the inconsistency of a given PC matrix [11, 12, 29]. In our case, there is only one triad hence:

$$\begin{pmatrix} 1 & x & y \\ 1/x & 1 & z \\ 1/y & 1/z & 1 \end{pmatrix}_{3\times 3}$$

The PCM above is determined by triad (x, y, z) and Kii is calculated using the formula: Kii(x, y, z) = 1 - min(x * z / y, y / (x * z)) where (x, y, z) is the only triad when the number of column or row in a PCM is three.

Triad is a vector consisting of three elements selected according to specific rules in the PC matrix. For PC matrices with rows or columns equal to 3, there is only one triad. Obviously, for the convenience of subsequent computation, we need to extract "triad" entities into a Java class which is shown in figure 15 below:

```java
package matrix_tool;

/**
 * Triad class.
 *
 * @author Yuqing Duan
 * @date 2018/10/22
 */
public class Triad {
    private double i;
    private double j;
    private double k;

    public Triad() {
        super();
    }

    public Triad(double i, double j, double k) {
        this.i = i;
        this.j = j;
        this.k = k;
    }

    public double getI() {
        return i;
    }

    public void setI(double i) {
        this.i = i;
    }

    public double getJ() {
        return j;
    }

    public void setJ(double j) {
        this.j = j;
    }

    public double getK() {
        return k;
    }

    public void setK(double k) {
        this.k = k;
    }

    @Override
    public String toString() {
        return "Triad{" +
                "i=" + i +
                ", j=" + j +
                ", k=" + k +
                '}';
    }
}
```

Figure 15: Triad Class

```java
/**
 * Calculates the Kii(Koczkodaj Inconsistency Indicator).
 *
 * @param triad triad vector for creating PC matrix
 * @return the value of Kii
 */
public static double computeKii(double[] triad) {
    BigDecimal w1 = new BigDecimal(String.valueOf(triad[0]));
    BigDecimal w2 = new BigDecimal(String.valueOf(triad[1]));
    BigDecimal w3 = new BigDecimal(String.valueOf(triad[2]));

    BigDecimal x = w1.divide(w2, scale: 15, BigDecimal.ROUND_HALF_UP);
    BigDecimal y = w1.divide(w3, scale: 15, BigDecimal.ROUND_HALF_UP);
    BigDecimal z = w2.divide(w3, scale: 15, BigDecimal.ROUND_HALF_UP);

    BigDecimal min1 = x.multiply(z).divide(y, scale: 15, BigDecimal.ROUND_HALF_UP);
    BigDecimal min2 = y.divide(x.multiply(z), scale: 15, BigDecimal.ROUND_HALF_UP);
    BigDecimal min;

    if (min1.compareTo(min2) == 1) {
        min = min2;
    } else {
        min = min1;
    }

    BigDecimal base = new BigDecimal( val: "1.0");
    BigDecimal sub = base.subtract(min);
    double Kii = sub.doubleValue();
    Kii = (double) Math.round(Kii * 100000) / 100000;
    return Kii;
}
```

Figure 16: computeKii() Function

Besides, figure 16 above is the java implementation of Kii calculation for a give n-dimensional PC matrix. Running the test code block "testDE.java", the following results can be obtained:

The test results for case 4:

Matrix M:

1.00000, 2.00000, 5.00000

0.50000, 1.00000, 3.00000

0.20000, 0.33333, 1.00000

The initial vector(EV/GM) is: (0.58155, 0.30900, 0.10945)

01(0.47818, 0.38646, 0.13536); Kii: 0.00000;

02(0.47818, 0.38646, 0.13536); Kii: 0.00000;

03(0.47818, 0.38646, 0.13536); Kii: 0.00000;

04(0.57990, 0.31539, 0.10471); Kii: 0.00000;

05(0.57873, 0.30180, 0.11947); Kii: 0.00000;

06(0.57873, 0.30180, 0.11947); Kii: 0.00000;

07(0.57873, 0.30180, 0.11947); Kii: 0.00000;

08(0.59059, 0.29792, 0.11148); Kii: 0.00000;

09(0.59059, 0.29792, 0.11148); Kii: 0.00000;

10(0.59059, 0.29792, 0.11148); Kii: 0.00000;

11(0.59059, 0.29792, 0.11148); Kii: 0.00000;

12(0.55075, 0.33777, 0.11148); Kii: 0.00000;

13(0.56415, 0.32150, 0.11435); Kii: 0.00000;

14(0.56740, 0.32129, 0.11131); Kii: 0.00000;

15(0.56740, 0.32129, 0.11131); Kii: 0.00000;

16(0.56740, 0.32129, 0.11131); Kii: 0.00000;

17(0.56740, 0.32129, 0.11131); Kii: 0.00000;

18(0.56740, 0.32129, 0.11131); Kii: 0.00000;

19(0.56740, 0.32129, 0.11131); Kii: 0.00000;

20(0.56740, 0.32129, 0.11131); Kii: 0.00000;

21(0.56740, 0.32129, 0.11131); Kii: 0.00000;

22(0.56740, 0.32129, 0.11131); Kii: 0.00000;

23(0.56740, 0.32129, 0.11131); Kii: 0.00000;

24(0.56740, 0.32129, 0.11131); Kii: 0.00000;

25(0.56740, 0.32129, 0.11131); Kii: 0.00000;

26(0.56740, 0.32129, 0.11131); Kii: 0.00000;

27(0.56740, 0.32129, 0.11131); Kii: 0.00000;

28(0.56889, 0.31950, 0.11161); Kii: 0.00000;

29(0.56889, 0.31950, 0.11161); Kii: 0.00000;

30(0.56889, 0.31950, 0.11161); Kii: 0.00000;

31(0.56889, 0.31950, 0.11161); Kii: 0.00000;

32(0.56889, 0.31950, 0.11161); Kii: 0.00000;

33(0.56889, 0.31950, 0.11161); Kii: 0.00000;

34(0.56889, 0.31950, 0.11161); Kii: 0.00000;

35(0.56914, 0.31905, 0.11181); Kii: 0.00000;

36(0.56914, 0.31905, 0.11181); Kii: 0.00000;

37(0.56914, 0.31905, 0.11181); Kii: 0.00000;

38(0.56914, 0.31905, 0.11181); Kii: 0.00000;

39(0.56914, 0.31905, 0.11181); Kii: 0.00000;

40(0.56914, 0.31905, 0.11181); Kii: 0.00000;

41(0.56914, 0.31905, 0.11181); Kii: 0.00000;

42(0.56914, 0.31905, 0.11181); Kii: 0.00000;

43(0.56914, 0.31905, 0.11181); Kii: 0.00000;

44(0.56914, 0.31905, 0.11181); Kii: 0.00000;

45(0.56914, 0.31905, 0.11181); Kii: 0.00000;

46(0.56914, 0.31905, 0.11181); Kii: 0.00000;

47(0.56914, 0.31905, 0.11181); Kii: 0.00000;

48(0.56914, 0.31905, 0.11181); Kii: 0.00000;

49(0.56914, 0.31905, 0.11181); Kii: 0.00000;

50(0.56914, 0.31905, 0.11181); Kii: 0.00000;

...

490(0.56880, 0.31930, 0.11190); Kii: 0.00000;

491(0.56880, 0.31930, 0.11190); Kii: 0.00000;

492(0.56880, 0.31930, 0.11190); Kii: 0.00000;

493(0.56880, 0.31930, 0.11190); Kii: 0.00000;

494(0.56880, 0.31930, 0.11190); Kii: 0.00000;

495(0.56880, 0.31930, 0.11190); Kii: 0.00000;

496(0.56880, 0.31930, 0.11190); Kii: 0.00000;

497(0.56880, 0.31930, 0.11190); Kii: 0.00000;

498(0.56880, 0.31930, 0.11190); Kii: 0.00000;

499(0.56880, 0.31930, 0.11190); Kii: 0.00000;

500(0.56880, 0.31930, 0.11190); Kii: 0.00000;

Result:

1. totalTriads: 1

2. Triad for initial M:

x1: 2.00000 y1: 5.00000 z1: 3.00000

3. Triad for evolved M:

x: 1.78138

y: 5.08326

z: 2.85355

4. Kii for initial M: 0.16667

5. Kii for evolved M: 0.00000

6. initial M: (0.58155, 0.30900, 0.10945)

7. evolved M: (0.56880, 0.31930, 0.11190)

However, when x is larger than three. There is more than one triad in the PC matrices. So we should find a way to get all the triads in a PC matrix. Also, store all the Kii value into an array. Then find the maximum Kii value in the Kii array.

The size of the array is immutable, and we cannot determine how many triads are in a given PC matrix. It is a good idea to use ArrayList as a container to store all the triads. After analysis, we need to adding another two functions in the project:

- Function1: findAllTriads(...)

- Function2: findMaxKii(...)

Function1 is the hardest to code. First, we need to analyze, find the law among the indexes of x, y, and z in a triad. Using a sketch map is more comfortable to analyze and illustrate.

1		(1,3)				(1,7)
	1		[2,4]		[2,6]	
		1				(3,7)
			1	{4,5}	[4,6]	{4,7}
				1		{5,7}
					1	
						1

Figure 17: PC Matrix with Various Triads

Figure 17 [27] above shows three triads composed of some of the elements in the PC matrix, which may not be adjacent elements in the matrix. Each triad uses different types of parentheses, just for demonstration purposes. All triads above the main diagonal line have a carpenter's corner shape or a capital letter "L" mirror image, with the "middle" value which is located in the "elbow" element ideally (for consistency) being the product of external elements.

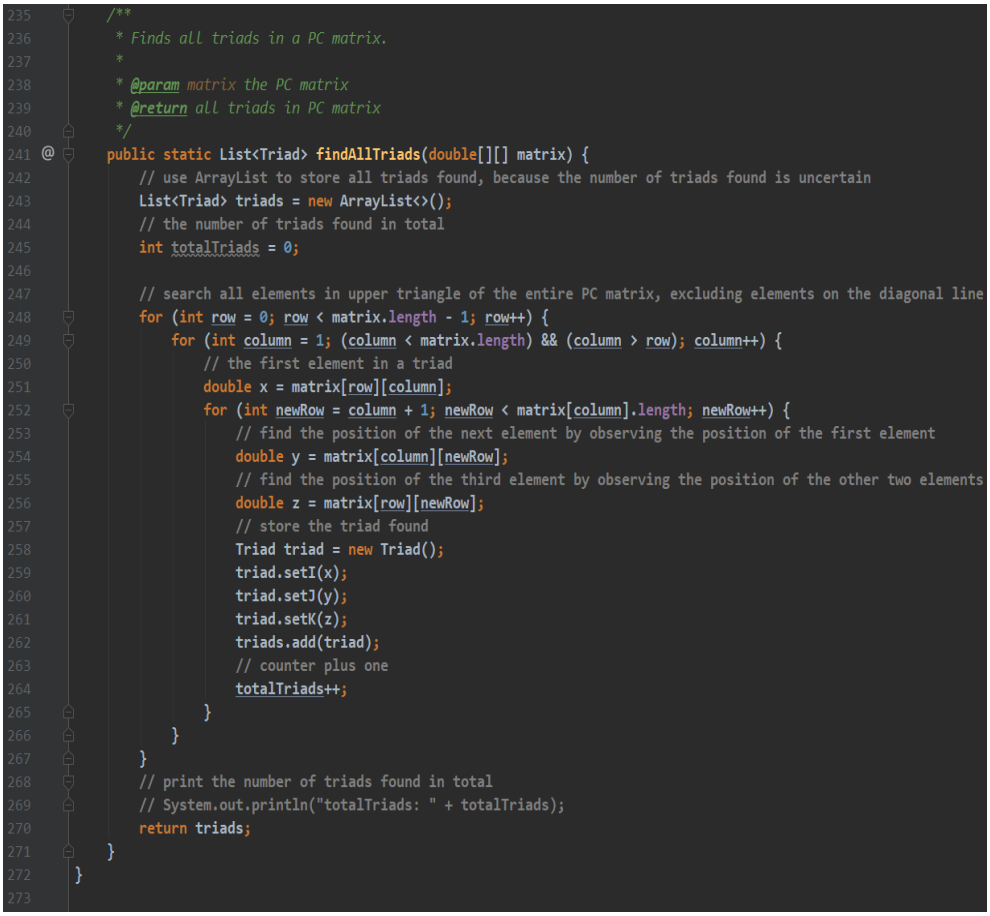
For each triad, the elements are not unique. That is to say, elements in one triad can appear in another triad. This situation is sometimes referred

to as overlapping, as illustrated in the following figure 18 [27].

1	(1,2)	(1,3)				
	1	(2,3)	(2,4)			
		1	(3,4)	(3,5)		
			1	(4,5)	(4,6)	
				1	(5,6)	(5,7)
					1	(6,7)
						1

Figure 18: Overlapping

For PC matrices with different dimensions, because the number of triads is uncertain (the larger the dimension, the more the number of triads), sets instead of arrays should be used to store all triads entities we can find in a given PC matrix. Firstly, searching all elements in the upper triangle of the entire PC matrix, excluding elements on the diagonal line. Secondly, for each element that is traversed, we consider it x (the first element in a triad). The position of the next element can be found by observing the position of the first element. Thirdly, doing the same operation and following the same rules to finding the last element of a triad in the given PC matrix. The relevant part of the code is shown in figure 19:



```java
/**
 * Finds all triads in a PC matrix.
 *
 * @param matrix the PC matrix
 * @return all triads in PC matrix
 */
public static List<Triad> findAllTriads(double[][] matrix) {
    // use ArrayList to store all triads found, because the number of triads found is uncertain
    List<Triad> triads = new ArrayList<>();
    // the number of triads found in total
    int totalTriads = 0;

    // search all elements in upper triangle of the entire PC matrix, excluding elements on the diagonal line
    for (int row = 0; row < matrix.length - 1; row++) {
        for (int column = 1; (column < matrix.length) && (column > row); column++) {
            // the first element in a triad
            double x = matrix[row][column];
            for (int newRow = column + 1; newRow < matrix[column].length; newRow++) {
                // find the position of the next element by observing the position of the first element
                double y = matrix[column][newRow];
                // find the position of the third element by observing the position of the other two elements
                double z = matrix[row][newRow];
                // store the triad found
                Triad triad = new Triad();
                triad.setI(x);
                triad.setJ(y);
                triad.setK(z);
                triads.add(triad);
                // counter plus one
                totalTriads++;
            }
        }
    }
    // print the number of triads found in total
    // System.out.println("totalTriads: " + totalTriads);
    return triads;
}
}
```

Figure 19: findAllTriads() Function

Each found triad correspondence can calculate a Kii value. In the experiment, for example, we take the PC matrix with dimension four as an example and find three different triad values. Correspondingly, three Kii values can be obtained, and then sort these values. We need to use the maximum Kii value as the criterion to measure the inconsistency of a given PC matrix. The way to find the maximum Kii is shown in figure 20:

```
105         *
106         * @param triads all triads in a given PC matrix
107         * @return maximum value of Kii
108         */
109 @       public static double findMaxKii(List<Triad> triads) {
110             // all Kii values
111             double[] kiiValues = new double[triads.size()];
112             int ind = 0;
113             // maximum Kii value
114             double maxKiiValue = 0;
115 
116             // traverse all triads
117             for (Triad t : triads) {
118                 // use BigDecimal class to improve the accuracy of calculation
119                 BigDecimal x = new BigDecimal(String.valueOf(t.getI()));
120                 BigDecimal z = new BigDecimal(String.valueOf(t.getJ()));
121                 BigDecimal y = new BigDecimal(String.valueOf(t.getK()));
122 
123                 // System.out.println("x: " + x);
124                 // System.out.println("y: " + y);
125                 // System.out.println("z: " + z);
126 
127                 // calculate 1 - min [ (y/x*z), (x*z/y) ]
128                 BigDecimal min1 = x.multiply(z).divide(y, scale: 15, BigDecimal.ROUND_HALF_UP);
129                 BigDecimal min2 = y.divide(x.multiply(z), scale: 15, BigDecimal.ROUND_HALF_UP);
130                 BigDecimal min;
131 
132                 if (min1.compareTo(min2) == 1) {
133                     min = min2;
134                 } else {
135                     min = min1;
136                 }
137 
138                 BigDecimal base = new BigDecimal( val: "1.0");
139                 BigDecimal sub = base.subtract(min);
140                 double kiiValue = sub.doubleValue();
141                 // kiiValue = (double) Math.round(kiiValue * 100000) / 100000;
142                 kiiValues[ind++] = kiiValue;
143             }
144 
145             // the maximum value among { 1 - min [ (y/x*z), (x*z/y) ] }
146             for (int x = 0; x < kiiValues.length; x++) {
147                 if (kiiValues[x] > maxKiiValue) {
148                     maxKiiValue = kiiValues[x];
149                 }
150             }
151             return maxKiiValue;
```

Figure 20: findMaxKii() Function

Using the functions we have created above as the evolution function for

the DE algorithm. The best individuals in each generation can be obtained. This time the number of row or column is four instead of three.

The test results for case 5:

Matrix M($x = 4$):

1.00000, 2.00000, 3.00000, 4.00000

0.50000, 1.00000, 5.00000, 6.00000

0.33333, 0.20000, 1.00000, 7.00000

0.25000, 0.16666, 0.14286, 1.00000

The initial vector(EV) is: (0.31283,0.35716,0.20997,0.12003)

01(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

02(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

03(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

04(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

05(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

06(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

07(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

08(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

09(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

10(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

11(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

12(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

13(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

14(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

15(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

16(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

17(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

18(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

19(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

20(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

21(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

22(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

23(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

24(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

25(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

26(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

27(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

28(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

29(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

30(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

31(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

32(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

33(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

34(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

35(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

36(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

37(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

38(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

39(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

40(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

41(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

42(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

43(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

44(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

45(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

46(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

47(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

48(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

49(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

50(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

...

490(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

491(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

492(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

493(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

494(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

495(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

496(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

497(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

498(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

499(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

500(0.17513, 0.31637, 0.29011, 0.21839); maxKii: 0.00000;

Result:

1. totalTriads: 3

2. Triad for initial M:

x1: 2.0 y1: 3.0 z1: 5.0;

x2: 2.0 y2: 4.0 z2: 6.0;

x3: 3.0 y3: 4.0 z3: 7.0.

3. Triad for evolved M:

x1: 1.9999439 y1: 1.0317805 z1: 0.5159047;

x2: 1.9999439 y2: 1.6778309 z2: 0.8389390;

x3: 1.0317805 y3: 1.6778309 z3: 1.6261511

4. Kii for initial M: 0.80952

5. Kii for evolved M: 0.00000

6. initial M: (0.31283, 0.35716, 0.20997, 0.12003)

7. evolved M: (0.17513, 0.31637, 0.29011, 0.21839)

8 Experiment on Corner PCM

1.Initial vector = GM

X	GM	DE	maxRE
1.0	(0.25000, 0.25000, 0.25000, 0.25000)	2.1373482705040265E-7	9.7472793170183 54E-8
1.1	(0.25599, 0.24996, 0.24996, 0.24408)	0.09556569763688852	0.032280434124253535
1.2	(0.26152, 0.24987, 0.24987, 0.23874)	0.1841029413192577	0.06265877521188443
1.3	(0.26666, 0.24973, 0.24973, 0.23388)	0.2676409537671763	0.09139304696514072
1.4	(0.27146, 0.24956, 0.24956, 0.22942)	0.34751944273088414	0.11868955440220952
1.5	(0.27596, 0.24936, 0.24936, 0.22532)	0.4246304455649132	0.14471450010741238
2.0	(0.29508, 0.24813, 0.24813, 0.20865)	0.7838911431638977	0.2599213210200748
3.0	(0.32289, 0.24534, 0.24534, 0.18642)	1.4187462134858928	0.4422498457782904
4.0	(0.34315, 0.24264, 0.24264, 0.17157)	1.9659775571646085	0.5874015928371092
5.0	(0.35911, 0.24015, 0.24015, 0.16060)	2.447244728326782	0.7099764792781742
6.0	(0.37228, 0.23787, 0.23787, 0.15198)	2.8803376476162468	0.8171211259128717
7.0	(0.38350, 0.23577, 0.23577, 0.14495)	3.277126751550058	0.9129316359298822
8.0	(0.39327, 0.23384, 0.23384, 0.13904)	3.6454685955732566	1.000000621061576
9.0	(0.40192, 0.23205, 0.23205, 0.13397)	3.990774667923108	1.080084307031696
10.0	(0.40968, 0.23038, 0.23038, 0.12955)	4.316935191978273	1.154434873368229

Figure 21: GM as Initial Vector

2.Initial vector = EV

X	EV	DE	maxRE
1.0	(0.2500000000000000, 0.2500000000000000, 0.24999999999999994, 0.25000000000000017)	2.0834376992190 65E-7	5.6994711172819734E-8
1.1	(0.25603010104900537, 0.2499290377347653, 0.2499290377347653, 0.24411182348146404)	0.0955656976369209	0.032280351601111555
1.2	(0.2616704314836689, 0.2497403937606802, 0.24974039376068033, 0.2388487809949704 2)	0.18410294131933735	0.06265871551166091
1.3	(0.2669818449343283, 0.24946261495911565, 0.2494626149591157, 0.2340929251474402 6)	0.26764095376713687	0.09139310320982676
1.4	(0.2720109735599784, 0.24911657471995988, 0.24911657471995993, 0.2297558770001017 2)	0.347519442730933	0.11868915896177312
1.5	(0.27679435561652305, 0.24871784772692607, 0.24871784772692596, 0.225769948929625)	0.4246304455648695	0.14471461957006942
2.0	(0.2978804187402747, 0.2462661721677227, 0.2462661721677228, 0.2095872369242797 7)	0.7838911431638679	0.25992126418307904
3.0	(0.3310079618533729, 0.24070280011042014, 0.24070280011042008, 0.1875864379257868 3)	1.4187462134859528	0.4422501306428894
4.0	(0.3570006158289384, 0.23533699070969277, 0.23533699070969297, 0.1723254027516758 2)	1.965977557164634	0.5874012748741746
5.0	(0.37849799942155754, 0.23042474855116954, 0.2304247485511696, 0.16065250347610324)	2.4472447283267575	0.7099762233682894
6.0	(0.3968349483600521, 0.22596339879437968, 0.22596339879437974, 0.1512382540511886)	2.880337647616295	0.8171212350634665
7.0	(0.41280879486412, 0.22190281083082705, 0.22190281083082714, 0.1433855834742259)	3.2771267515500115	0.912931763864379
8.0	(0.42694253745835564, 0.21818913599623804, 0.21818913599623813, 0.1366791905491681 3)	3.6454685955732726	1.0000003869013683
9.0	(0.43960075340429333, 0.2147745841221857, 0.21477458412218564, 0.1308500783513353 6)	3.9907746679231155	1.0800840508154093
10.0	(0.451048860363806, 0.21161884485276847, 0.21161884485276855, 0.1257134499306569 2)	4.316935191978287	1.1544352961863869

Figure 22: EV as Initial Vector

Both figure 21 and figure 22 show the test results when using the DE algorithm for a particular kind of PC matrix, and the corner PC matrix used for the test is as follows:

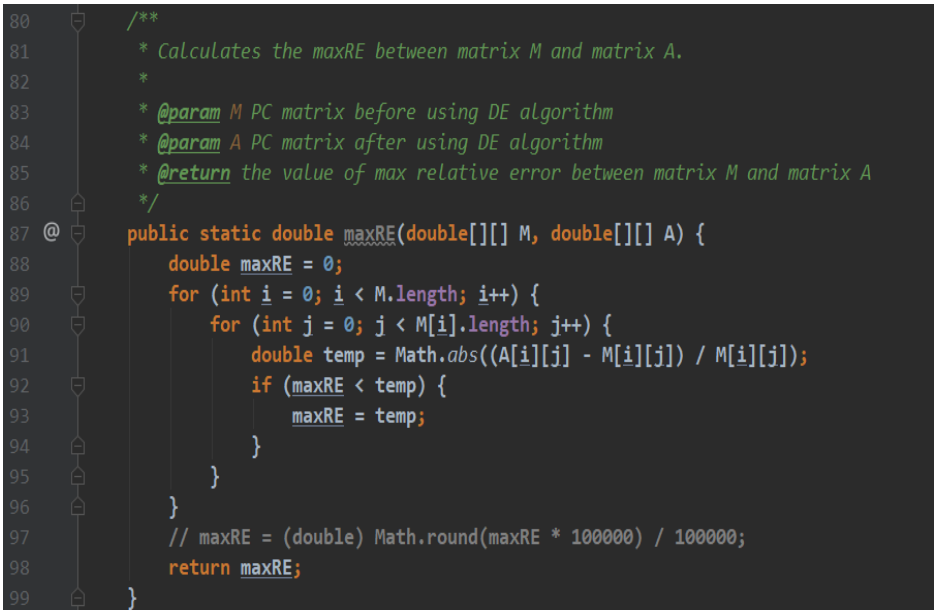
$$\begin{pmatrix} 1 & 1 & 1 & x \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1/x & 1 & 1 & 1 \end{pmatrix}_{4\times 4}$$

What can be acquired from the picture is that regardless of which kind of input (GM or EV), both ED and maxRE are increasing when x is increasing. However, the speeds of growth are different. ED increases faster than maxRE. For each group, the maxRE are always smaller than EDs.

It is worth noting that maxRE is a little bit larger than 1 when x equals eight, nine and ten. Checking the relative error formula and analyzing this phenomenon:

$$\eta = \frac{\epsilon}{|v|} = \left|\frac{v - v_{\text{approx}}}{v}\right| = \left|1 - \frac{v_{\text{approx}}}{v}\right|,$$

If x increases, the inconsistent matrix will become more complex and changeable. As a result, the new individuals created by DE algorithm each generation will go far from the initial one. The ratio value of V_{approx}/v will increase if we raise the value of x. So it is not difficult to comprehend that maxRE will beyond 1 when x is getting larger and larger. If we track the execution process of the code or debug the code below:



```java
/**
 * Calculates the maxRE between matrix M and matrix A.
 *
 * @param M PC matrix before using DE algorithm
 * @param A PC matrix after using DE algorithm
 * @return the value of max relative error between matrix M and matrix A
 */
public static double maxRE(double[][] M, double[][] A) {
    double maxRE = 0;
    for (int i = 0; i < M.length; i++) {
        for (int j = 0; j < M[i].length; j++) {
            double temp = Math.abs((A[i][j] - M[i][j]) / M[i][j]);
            if (maxRE < temp) {
                maxRE = temp;
            }
        }
    }
    // maxRE = (double) Math.round(maxRE * 100000) / 100000;
    return maxRE;
}
```

Figure 23: maxRE() Function

Getting all the values for A[i][j], M[i][j] and maxRE each generation, which can be helpful to verify the conclusions we obtained in former paragraph. In other words, we can see directly that when x increase, the overall level of values for Math.abs((A[i][j] - M[i][j]) / M[i][j]) will increase in each iteration. At the same time $1/x$ is getting lower. Therefore, maxRE will increase.

9 Creation of Fully Consistent PC Matrices

9.1 Codes and Annotations

```java
/**
 * Random generation of PC matrices.
 *
 * @param n the number of rows and columns of initial PC matrix
 * @return an n*n two-dimensional array.
 */
public static double[][] createPCMatrix(int n) {

    double[] v = new double[n];
    double[][] M = new double[n][n];
    // change the value from 0.75000 to 1.00000
    // when Δ=0.23000 (minThreshhold=0.77000 and maxThreshhold=3.23000) trunRate begin to above 10.000%
    double minThreshhold = 0.85000;
    // change the value from 3.00000 to 3.25000
    double maxThreshhold = 3.15000;

    // create the random vector v_1, v_2 ...v_n
    for (int k = 0; k < v.length; k++) {
        Random r = new Random();
        v[k] = r.nextDouble();
    }

    // "vector filter" code block
    // sort the specified range of the array into ascending order
    Arrays.sort(v);
    // v[v.length - 1] is the maximum element in vector, and v[0] is the minimum element in vector
    double maxQuotient = v[v.length - 1] / v[0];
    while ((maxQuotient < minThreshhold) || (maxQuotient > maxThreshhold)) {
        // recalculate the maximum quotient
        if ((1.0 / v[0]) > (v[v.length - 1])) {
            // change v[0] to v[0] * 2.0 is a better choice in this case
            Arrays.fill(v, fromIndex: 0, toIndex: 1, (v[0] * 2.0));
        } else {
            // change v[v.length - 1] to v[v.length - 1] / 2.0 is a better choice in this case
            Arrays.fill(v, fromIndex: v.length - 1, v.length, (v[v.length - 1] / 2.0));
        }
        // resort the vector
        Arrays.sort(v);
        maxQuotient = v[v.length - 1] / v[0];
    }

    // use the random vector to construct a consistent PC matrix M
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            M[i][j] = v[i] / v[j];
        }
    }
    return M;
}
}
```

Figure 24: createPCMatrix() Function

In createPCMatrix() function, we need to create 1000 samples of fully consistent PC matrix for $x = 3, 4, 5, ..., 7$ separately, and the values of all elements in each PC matrix should between 0.33333 to 3.00000. A vector filter was utilized to let all values in that range.

9.2 Result Samples

The test program generated 6000 fully consistent PC matrices (For the test program, please check appendix: TestMatrixInitialization.java).

$x = 3$

1.00000, 0.85518, 0.40019

1.16934, 1.00000, 0.46796

2.49882, 2.13695, 1.00000

$x = 4$

1.00000, 0.98031, 0.36170, 0.35567

1.02008, 1.00000, 0.36897, 0.36282

2.76471, 2.71028, 1.00000, 0.98333

2.81156, 2.75622, 1.01695, 1.00000

$x = 5$

1.00000, 0.73047, 0.62815, 0.44059, 0.38104

1.36898, 1.00000, 0.85992, 0.60316, 0.52164

1.59198, 1.16290, 1.00000, 0.70141, 0.60661

2.26969, 1.65794, 1.42570, 1.00000, 0.86485

2.62438, 1.91703, 1.64849, 1.15627, 1.00000

———————————————————-

$x = 6$

1.00000, 0.94546, 0.82352, 0.75444, 0.66723, 0.42007

1.05769, 1.00000, 0.87103, 0.79796, 0.70573, 0.44430

1.21431, 1.14807, 1.00000, 0.91612, 0.81023, 0.51009

1.32549, 1.25319, 1.09156, 1.00000, 0.88441, 0.55680

1.49873, 1.41698, 1.23422, 1.13070, 1.00000, 0.62957

2.38055, 2.25071, 1.96042, 1.79598, 1.58839, 1.00000

———————————————————-

$x = 7$

1.00000, 0.94032, 0.79195, 0.59263, 0.51387, 0.50008, 0.34280

1.06347, 1.00000, 0.84222, 0.63024, 0.54649, 0.53182, 0.36456

1.26270, 1.18734, 1.00000, 0.74832, 0.64887, 0.63145, 0.43285

1.68739, 1.58669, 1.33633, 1.00000, 0.86711, 0.84383, 0.57844

1.94600, 1.82986, 1.54114, 1.15326, 1.00000, 0.97315, 0.66709

1.99969, 1.88034, 1.58365, 1.18507, 1.02759, 1.00000, 0.68549

2.91715, 2.74305, 2.31025, 1.72879, 1.49905, 1.45881, 1.00000

9.3 Truncate Rate and Total Time Used

As for the code in line 23 and line 27 in figure24, different threshold values were set to check the changes in the truncate rate and total time used in creating whole matrices. The result is shown in figure 25 and 26 below:

	A	B	C	D	E	F
1	Experiment items / Experiment No.	minThreshhold	maxThreshhold	totalRandomM	trunCounter	trunRate
2	1	1.00000	3.00000	1000	0	0.000%
3	2	1.00000	3.00000	1000	0	0.000%
4	3	1.00000	3.00000	1000	0	0.000%
5	4	1.00000	3.00000	1000	0	0.000%
6	5	1.00000	3.00000	1000	0	0.000%
7	6	0.95000	3.05000	1000	27	2.700%
8	7	0.95000	3.05000	1000	22	2.200%
9	8	0.95000	3.05000	1000	27	2.700%
10	9	0.95000	3.05000	1000	19	1.900%
11	10	0.95000	3.05000	1000	23	2.300%
12	11	0.90000	3.10000	1000	41	4.100%
13	12	0.90000	3.10000	1000	44	4.400%
14	13	0.90000	3.10000	1000	53	5.300%
15	14	0.90000	3.10000	1000	38	3.800%
16	15	0.90000	3.10000	1000	40	4.000%
17	16	0.85000	3.15000	1000	70	7.000%
18	17	0.85000	3.15000	1000	77	7.700%
19	18	0.85000	3.15000	1000	73	7.300%
20	19	0.85000	3.15000	1000	68	6.800%
21	20	0.85000	3.15000	1000	59	5.900%
22	21	0.80000	3.20000	1000	84	8.400%
23	22	0.80000	3.20000	1000	87	8.700%
24	23	0.80000	3.20000	1000	83	8.300%
25	24	0.80000	3.20000	1000	82	8.200%
26	25	0.80000	3.20000	1000	95	9.500%
27	26	0.75000	3.25000	1000	109	10.900%
28	27	0.75000	3.25000	1000	93	9.300%
29	28	0.75000	3.25000	1000	102	10.200%
30	29	0.75000	3.25000	1000	92	9.200%
31	30	0.75000	3.25000	1000	96	9.600%

Figure 25: The Relationship between Threshold and Totaltime

The figure 25 shows that truncate rate increases with the increase of the difference between minThreshhold and maxThreshhold, this increase is not

strict, but generally, we can see that the truncate rate increases synchronously with the threshold difference. It is worth noting that when $\delta = 0.23000$ (minThreshhold=0.77000 and maxThreshhold=3.23000) trunRate begin to above 10.000, which means the truncate rate is a little bit higher.

	A	B	C	D	E
1	Experiment items / Experiment No.(i±0.01)	totalRandomM	trunCounter	trunRate	totalTime (ms)
2	1	10000	0	0.000%	2033
3	2	10000	46	0.460%	1953
4	3	10000	70	0.700%	2221
5	4	10000	129	1.290%	2031
6	5	10000	178	1.780%	1978
7	6	10000	220	2.200%	2259
8	7	10000	219	2.190%	2087
9	8	10000	303	3.030%	2068
10	9	10000	319	3.190%	2050
11	10	10000	380	3.800%	2321
12	11	10000	416	4.160%	2054
13	12	10000	510	5.100%	1929
14	13	10000	581	5.810%	1660
15	14	10000	611	6.110%	1565
16	15	10000	654	6.540%	1676
17	16	10000	715	7.150%	2174
18	17	10000	723	7.230%	2186
19	18	10000	756	7.560%	1698
20	19	10000	843	8.430%	1855
21	20	10000	887	8.870%	1681
22	21	10000	937	9.370%	1778
23	22	10000	994	9.940%	1757
24	23	10000	1011	10.110%	1697
25	24	10000	1074	10.740%	1987
26	25	10000	1097	10.970%	1766

Figure 26: The Relationship between Trunrate and Totaltime

The function currtTimeMillis() returns the current time in milliseconds and the time difference between the current time and the midnight of Harmonized World Time on January 1, 1970 (measured in milliseconds). Please

check figure 27 and figure 28 below. When the unit of time for the return value is milliseconds, the granularity of the value depends on the underlying operating system and may be larger.

We use this system function to calculate the running time of the program in order to improve accuracy. The total number of generated samples is increased to 10,000 this time.

```
42          // the start time for the code block
43          long startTime = System.currentTimeMillis();
```

Figure 27: The start time for the code block

```
216         // the end time for the code block
217         long endTime = System.currentTimeMillis();
```

Figure 28: The end time for the code block

10 NSI PC Matrix

It is not only necessary to do deviation operation on an initially given PC matrix, but also to ensure that the matrix can not deviate too far from its original appearance. That is to say; we are going to construct a kind of PC matrix called NSI (Not So Inconsistent) PC matrix [26].

10.1 The Necessity of Deviation

As the results (page 91 to 93) had shown us: the value of Kii is always equals zero. Then a new tool class java.Bigdecimal was utilized to improve the

accuracy of calculation, in order to make the value of Kii not always equals zero. However, we failed to get the expected results.

It is easy to find that the problem is that $min(1.000002663, 0.999997337)$ is very close to one after debugging and tracking the code. So the result of Kii will very close to zero. That is why we try to improve precision when calculating Kii in each iteration. However, it still always be $Kii = 0$ in each generation. We also check if $abs(1 - Kii) < eps$ where "abs" is to calculate the absolute value of $1 - Kii$ and $eps = 0.001$ and then stop. The code is as follows:

```
private double scorefunc2(double[] individual) {
   double sco = 0.0;
   sco = EvaluationFunction.computeKii(individual);
   if (Math.abs(1 - sco) >= 0.001) {
      return -sco;
   } else {
      return -1;
   }
}
```

It did not work. Let us analyze the Kii process using fundamental mathematics knowledge: If Kii is always very close to 0, that means $1 - Kii$ always very close to 1. So the code can not meet the stop condition forever.

Therefore, we need to deviate values of the elements in PC matrices which

are created by the function: computeMatrixA(double[] vec) after each iteration. The original computeMatrixA(double[] vec) function shown below is what we use to reconstruct PC matrix each iteration:

```
75      /**
76       * Creates PC matrix by follow rules: A_ij = v_i/v_j where initially vector v = GM and is improved by DE algorithm.
77       *
78       * @param vec vector used to create PC matrix
79       * @return PC matrix after using DE algorithm
80       */
81  @   public static double[][] computeMatrixA(double[] vec) {
82          double[][] A = new double[vec.length][vec.length];
83          for (int i = 0; i < A.length; i++) {
84              for (int j = 0; j < A[i].length; j++) {
85                  A[i][j] = Math.abs(vec[i] / vec[j]);
86              }
87          }
88          return A;
89      }
```

Figure 29: computeMatrixA() Function

10.2 Some Wrong Experimental Code and Results:

The deviation rate is +20%, and only the elements above the main diagonal of PC matrices deviated. The corresponding code block I created for deviating PC matrices is shown as follows:

```java
public static double[][] computeDeviatedMatrixA(double[] vec) {
    double[][] A = new double[vec.length][vec.length];
    for (int i = 0; i < A.length; i++) {
        for (int j = 0; j < A[i].length; j++) {
            A[i][j] = Math.abs(vec[i] / vec[j]);
            if (i < j) {
                // increase by 20%
                A[i][j] *= 1.20000;
            }
        }
    }
    return A;
}
```

Figure 30: computeDeviatedMatrixA() Function

The test results for case 6:

Matrix M($x = 4$):

1.00000, 2.00000, 3.00000, 4.00000

0.50000, 1.00000, 5.00000, 6.00000

0.33333, 0.20000, 1.00000, 7.00000

0.25000, 0.16666, 0.14286, 1.00000

The initial vector(EV) is: (0.31283, 0.35716, 0.20997, 0.12003);

01(0.16272, 0.15231, 0.31878, 0.36618); maxKii:0.166666666666667;

02(0.16272, 0.15231, 0.31878, 0.36618); maxKii:0.166666666666667;

03(0.16272, 0.15231, 0.31878, 0.36618); maxKii:0.166666666666667;

04(0.16272, 0.15231, 0.31878, 0.36618); maxKii:0.166666666666667;

05(0.14709, 0.08073, 0.18840, 0.58377); maxKii:0.166666666666666;

06(0.14709, 0.08073, 0.18840, 0.58377); maxKii:0.166666666666666;

07(0.14709, 0.08073, 0.18840, 0.58377); maxKii:0.166666666666666;

...

93(0.14709, 0.08073, 0.18840, 0.58377); maxKii: 0.166666666666666;

94(0.14709, 0.08073, 0.18840, 0.58377); maxKii: 0.166666666666666;

95(0.14709, 0.08073, 0.18840, 0.58377); maxKii: 0.166666666666666;

96(0.30854, 0.38392, 0.08087, 0.22667); maxKii: 0.166666666666666;

97(0.30854, 0.38392, 0.08087, 0.22667); maxKii: 0.166666666666666;

98(0.30854, 0.38392, 0.08087, 0.22667); maxKii: 0.166666666666666;

...

158(0.30854, 0.38392, 0.08087, 0.22667); maxKii:0.166666666666666;

159(0.30854, 0.38392, 0.08087, 0.22667); maxKii:0.166666666666666;

160(0.30854, 0.38392, 0.08087, 0.22667); maxKii:0.166666666666666;

161(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

162(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

163(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

...

192(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

193(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

194(0.51132, 0.23658, 0.20838, 0.04373); maxKii:0.166666666666666;

195(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

196(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

197(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

...

498(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

499(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

500(0.36585, 0.42899, 0.01380, 0.19136); maxKii:0.166666666666666;

1. maxKii of initial M: 0.80952380952381
2. maxKii of evolved M: 0.166666666666666
3. initial M: (0.31283, 0.35716, 0.20997, 0.12003)
4. evolved M: (0.36585, 0.42899, 0.01380, 0.19136)

10.3 Reflections on The Wrong Results in Section 10.2

For the elements above the main diagonal line in the PC matrix, we can not apply the same fluctuation range (for example, the element values are increased by 20 percent on the original basis) to them.

According to the relevant calculation formula, we can easily find that if we do this, we will get the wrong result: for all individuals in each generation, their maxKii value is precisely the same but may not zero, as shown in the previous experiment. This means that the genetic algorithm can not complete the selection operation to select the better individuals in each generation. One possible solution is to impose a different amplitude of fluctuation on the elements above the principal diagonal in the PC matrix. Please check the results in the experiment below (Gen99 and Gen100):

99(0.30290, 0.22053, 0.46340, 0.01317);

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

100(0.30290, 0.22053, 0.46340, 0.01317);

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

Score:-0.166666666666667

In the experiment, the population size was set as 10, i.e., there were ten

individuals in each generation. As can be seen, the functions of 10 individuals in the ninety-ninth generation have the same value as those in the one hundred generations.

First of all, applying the square operation to the target elements. It is not difficult to find that this will make the fluctuation range of the elements in different positions different. For example, if $A[0][2] = 3$, the new $A[0][2]$ will become 9, the element value will become 3 times $A[0][3] = 4$, the new $A[0][2]$ will become 16 and the element value will become 4 times of the original.

10.4 Modified Code and Results

Idea: substituting squares for multiplying a constant value:

```
147 @    public static double[][] computeDeviatedMatrixA(double[] vec) {
148          double[][] A = new double[vec.length][vec.length];
149          for (int i = 0; i < A.length; i++) {
150              for (int j = 0; j < A[i].length; j++) {
151                  A[i][j] = Math.abs(vec[i] * vec[i] / vec[j]);
152              }
153          }
154          return A;
155      }
```

The test results for case 7:

The initial vector(EV) is: (0.31283, 0.35716, 0.20997, 0.12003);

01(0.44763, 0.04004, 0.15418, 0.35815); maxKii: 0.685624755030971

02(0.44763, 0.04004, 0.15418, 0.35815); maxKii: 0.685624755030971

03(0.44763, 0.04004, 0.15418, 0.35815); maxKii: 0.685624755030971

04(0.44763, 0.04004, 0.15418, 0.35815); maxKii: 0.685624755030971

05(0.44763, 0.04004, 0.15418, 0.35815); maxKii: 0.685624755030971

06(0.01423, 0.12979, 0.17173, 0.68426); maxKii: 0.539393043689909

07(0.42862, 0.03834, 0.06983, 0.46321); maxKii: 0.335302528304303

08(0.42862, 0.03834, 0.06983, 0.46321); maxKii: 0.335302528304303

09(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

10(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

11(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

12(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

13(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

14(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

15(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

16(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

17(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

18(0.43238, 0.05786, 0.08941, 0.42035); maxKii: 0.262801963197629

19(0.27085, 0.04988, 0.07667, 0.60260); maxKii: 0.21248796836324

20(0.27085, 0.04988, 0.07667, 0.60260); maxKii: 0.21248796836324

21(0.36056, 0.05658, 0.07126, 0.51160); maxKii: 0.189598805259846

22(0.36056, 0.05658, 0.07126, 0.51160); maxKii: 0.189598805259846

23(0.35992, 0.04816, 0.06718, 0.52475); maxKii: 0.183225381455881

24(0.35992, 0.04816, 0.06718, 0.52475); maxKii: 0.183225381455881

25(0.35992, 0.04816, 0.06718, 0.52475); maxKii: 0.183225381455881

26(0.40910, 0.04455, 0.06441, 0.48195); maxKii: 0.175671937284222

27(0.40910, 0.04455, 0.06441, 0.48195); maxKii: 0.175671937284222

28(0.56196, 0.06616, 0.08764, 0.28423); maxKii: 0.167874339252076

29(0.34124, 0.05626, 0.07388, 0.52863); maxKii: 0.160589362208309

30(0.34124, 0.05626, 0.07388, 0.52863); maxKii: 0.160589362208309

31(0.40680, 0.05443, 0.07383, 0.46494); maxKii: 0.160029239975411

32(0.40680, 0.05443, 0.07383, 0.46494); maxKii: 0.160029239975411

33(0.40715, 0.05417, 0.07333, 0.46535); maxKii: 0.153618411212379

34(0.40715, 0.05417, 0.07333, 0.46535); maxKii: 0.153618411212379

35(0.37757, 0.05778, 0.07931, 0.48535); maxKii: 0.152900078352032

36(0.37757, 0.05778, 0.07931, 0.48535); maxKii: 0.152900078352032

37(0.38614, 0.06366, 0.08243, 0.46777); maxKii: 0.14863942440941

38(0.29127, 0.05579, 0.07222, 0.58073); maxKii: 0.143725684508491

39(0.39885, 0.06548, 0.08272, 0.45295); maxKii: 0.142072280422184

40(0.39885, 0.06548, 0.08272, 0.45295); maxKii: 0.142072280422184

41(0.39885, 0.06548, 0.08272, 0.45295); maxKii: 0.142072280422184

42(0.43584, 0.06951, 0.08654, 0.40811); maxKii: 0.138810678357457

43(0.37489, 0.06252, 0.08067, 0.48192); maxKii: 0.135861520414634

44(0.37489, 0.06252, 0.08067, 0.48192); maxKii: 0.135861520414634

45(0.37489, 0.06252, 0.08067, 0.48192); maxKii: 0.135861520414634

46(0.42931, 0.06686, 0.08853, 0.41530); maxKii: 0.134946524318291

47(0.29974, 0.05266, 0.06939, 0.57821); maxKii: 0.133522064972307

48(0.29974, 0.05266, 0.06939, 0.57821); maxKii: 0.133522064972307

49(0.32946, 0.05714, 0.07522, 0.53818); maxKii: 0.132245300991402

50(0.40837, 0.06234, 0.07806, 0.45123); maxKii: 0.132231131966077

51(0.40837, 0.06234, 0.07806, 0.45123); maxKii: 0.132231131966077

52(0.40837, 0.06234, 0.07806, 0.45123); maxKii: 0.132231131966077

53(0.43017, 0.06384, 0.08180, 0.42420); maxKii: 0.132164991148833

54(0.31494, 0.05783, 0.07429, 0.55295); maxKii: 0.131997657761473

55(0.35934, 0.05541, 0.06994, 0.51531); maxKii: 0.131698050364828

56(0.38957, 0.05535, 0.06896, 0.48612); maxKii: 0.131463693008229

57(0.38957, 0.05535, 0.06896, 0.48612); maxKii: 0.131463693008229

58(0.37857, 0.05182, 0.06623, 0.50338); maxKii: 0.13141168287344

59(0.39162, 0.06107, 0.07707, 0.47024); maxKii: 0.131327161954461

60(0.39162, 0.06107, 0.07707, 0.47024); maxKii: 0.131327161954461

61(0.39162, 0.06107, 0.07707, 0.47024); maxKii: 0.131327161954461

62(0.39162, 0.06107, 0.07707, 0.47024); maxKii: 0.131327161954461

63(0.37865, 0.05162, 0.06623, 0.50350); maxKii: 0.131264091771809

64(0.38656, 0.06281, 0.08037, 0.47025); maxKii: 0.131206947291262

65(0.38656, 0.06281, 0.08037, 0.47025); maxKii: 0.131206947291262

66(0.34686, 0.04972, 0.06294, 0.54047); maxKii: 0.13107486960102

67(0.36830, 0.05353, 0.06719, 0.51097); maxKii: 0.131032371239377

68(0.36830, 0.05353, 0.06719, 0.51097); maxKii: 0.131032371239377

69(0.36830, 0.05353, 0.06719, 0.51097); maxKii: 0.131032371239377

70(0.36830, 0.05353, 0.06719, 0.51097); maxKii: 0.131032371239377

...

490(0.66483, 0.08916, 0.11300, 0.13301); maxKii: 0.130469888817024

491(0.66483, 0.08916, 0.11300, 0.13301); maxKii: 0.130469888817024

492(0.64608, 0.09415, 0.11932, 0.14045); maxKii: 0.130468919413328

493(0.64608, 0.09415, 0.11932, 0.14045); maxKii: 0.130468919413328

494(0.58504, 0.08432, 0.10242, 0.22822); maxKii: 0.130440744127969

495(0.49526, 0.06645, 0.08398, 0.35431); maxKii: 0.130437482245267

496(0.49526, 0.06645, 0.08398, 0.35431); maxKii: 0.130437482245267

497(0.49526, 0.06645, 0.08398, 0.35431); maxKii: 0.130437482245267

498(0.64601, 0.09417, 0.11934, 0.14048); maxKii: 0.130432582468055

499(0.54491, 0.07589, 0.09549, 0.28372); maxKii: 0.130430239513515

500(0.59947, 0.08702, 0.10495, 0.20856); maxKii: 0.130421107513497

1. maxKii of initial M: 0.80952380952381
2. maxKii of evolved M: 0.130421107513497
3. initial M: (0.31283, 0.35716, 0.20997, 0.12003)
4. evolved M: (0.59947, 0.08702, 0.10495, 0.20856)

10.5 Further Discussion of the Results in Section 10.4

As is shown above, in this way, the maxKii values of different generations will inevitably be different. We select the candidate individuals with the lowest maxKii value from each generation to produce the next generation so that the whole population can complete its evolution.

Of course, we should try to improve the algorithm properly, because a

good result is evolved. M is not identical to the initial M but very close to it. Now we can think of some further improvements: cubic operation, square operation, trigonometric function operation and so on.

10.6 Explore The Correctness of Applying DE Algorithm to PC Matrices' Global Optimization

The results did not converge within 100 generations when I tried to compute the approximation PC matrix A for the specifically given PC matrix M: [[1.00000, 2.00000, 5.00000], [0.50000, 1.00000, 3.00000], [0.20000, 0.33333, 1.00000]], and the initial solution is a fully consistent PC matrix created by the geometric means (GM) vector.

In order to further explore the correctness of differential evolution (DE) algorithm applied to PC matrices global optimization, I increased the number of iteration to 1000. This time I only focused on observing the experimental results where changes contain after 100 generations and made necessary records of those items with "good evolution." Choosing ED, MED, and maxRE as evaluation function in the DE algorithm respectively:

% Chooses Euclidean Distance (ED) as evaluation function %

% population_size = 50; iter_num = 1000; param_bottom_bound = 0; param_upper_bound = 10; n = 3 %

% PCM = [[1.00000, 2.00000, 5.00000], [0.50000, 1.00000, 3.00000], [0.20000, 0.33333, 1.00000]] %

The initial vector is: (0.58155,0.30900,0.10945)

100(0.56879, 0.31931, 0.11190); ED: 0.28327177861186625

101(0.56880, 0.31930, 0.11190); ED: 0.28327177398634656

...

107(0.56880, 0.31930, 0.11190); ED: 0.283271759922282

...

111(0.56880, 0.31930, 0.11190); ED: 0.28327175219943496

...

129(0.56880, 0.31930, 0.11190); ED: 0.28327175124406

...

190(0.56880, 0.31930, 0.11190); ED: 0.28327175124261816

...

195(0.56880, 0.31930, 0.11190); ED: 0.28327175124171244

...

207(0.56880, 0.31930, 0.11190); ED: 0.28327175124164394

...

214(0.56880, 0.31930, 0.11190); ED: 0.28327175124160353

...

219(0.56880, 0.31930, 0.11190); ED: 0.2832717512414945

...

231(0.56880, 0.31930, 0.11190); ED: 0.2832717512414913

...

240(0.56880, 0.31930, 0.11190); ED: 0.28327175124148096

...

247(0.56880, 0.31930, 0.11190); ED: 0.28327175124147996

...

257(0.56880, 0.31930, 0.11190); ED: 0.28327175124147713

...

270(0.56880, 0.31930, 0.11190); ED: 0.2832717512414762

...

280(0.56880, 0.31930, 0.11190); ED: 0.28327175124147597

...

286(0.56880, 0.31930, 0.11190); ED: 0.2832717512414759

...

325(0.56880, 0.31930, 0.11190); ED: 0.28327175124147586

...

328(0.56880, 0.31930, 0.11190); ED: 0.2832717512414758

...

Result after 1000 iteration:

(0.56880, 0.31930, 0.11190)

% Chooses Modified Euclidean Distance (MED) as evaluation function %

% population_size = 50; iter_num = 1000; param_bottom_bound = 0; param_upper_bound = 10; n = 3 %

% PCM = [[1.00000, 2.00000, 5.00000], [0.50000, 1.00000, 3.00000], [0.20000,

0.33333, 1.00000]] %

The initial vector is: (0.58155,0.30900,0.10945)

100(0.56879, 0.31932, 0.11189); MED: 0.03147464986157849

...

111(0.56880, 0.31931, 0.11190); MED: 0.031474640441499856

...

122(0.56880, 0.31931, 0.11190); MED: 0.031474640070840915

...

125(0.56880, 0.31930, 0.11190); MED: 0.031474639061795234

...

150(0.56880, 0.31930, 0.11190); MED: 0.03147463903714041

151(0.56880, 0.31930, 0.11190); MED: 0.03147463902796486

...

165(0.56880, 0.31930, 0.11190); MED: 0.031474639027366004

...

171(0.56880, 0.31930, 0.11190); MED: 0.031474639027047446

...

181(0.56880, 0.31930, 0.11190); MED: 0.031474639026869956

...

187(0.56880, 0.31930, 0.11190); MED: 0.031474639026866785

...

192(0.56880, 0.31930, 0.11190); MED: 0.03147463902684712

...

201(0.56880, 0.31930, 0.11190); MED: 0.031474639026843006

...

203(0.56880, 0.31930, 0.11190); MED: 0.031474639026841694

...

205(0.56880, 0.31930, 0.11190); MED: 0.03147463902684147

...

210(0.56880, 0.31930, 0.11190); MED: 0.03147463902683193

...

212(0.56880, 0.31930, 0.11190); MED: 0.03147463902683069

...

249(0.56880, 0.31930, 0.11190); MED: 0.031474639026830675

...

256(0.56880, 0.31930, 0.11190); MED: 0.03147463902683066

...

267(0.56880, 0.31930, 0.11190); MED: 0.03147463902683065

...

Result after 1000 iteration:

(0.56880, 0.31930, 0.11190)

% Chooses Maximum Relative Error (maxRE) as evaluation function %

% population_size = 50; iter_num = 1000; param_bottom_bound = 0; param_upper_bound = 10; n = 3 %

% PCM = [[1.00000, 2.00000, 5.00000], [0.50000, 1.00000, 3.00000], [0.20000, 0.33333, 1.00000]] %

The initial vector is: (0.58155, 0.30900, 0.10945)

100(0.58156, 0.30899, 0.10945); maxRE: 0.06269289300834853

...

108(0.58155, 0.30900, 0.10945); maxRE: 0.06268511117344744

109(0.58156, 0.30899, 0.10945); maxRE: 0.06268476623837796

...

112(0.58155, 0.30900, 0.10945); maxRE: 0.06267064302058896

...

139(0.58155, 0.30900, 0.10945); maxRE: 0.06266887554877743

...

143(0.58155, 0.30900, 0.10945); maxRE: 0.0626667530060514

...

154(0.58155, 0.30900, 0.10945); maxRE: 0.06266634449997799

155(0.58155, 0.30900, 0.10945); maxRE: 0.06266590789311932

156(0.58155, 0.30900, 0.10945); maxRE: 0.06266404354529616

...

168(0.58155, 0.30900, 0.10945); maxRE: 0.06266347720002248

169(0.58155, 0.30900, 0.10945); maxRE: 0.06266329512072412

...

179(0.58155, 0.30900, 0.10945); maxRE: 0.0626628416931776

180(0.58155, 0.30900, 0.10945); maxRE: 0.06266280259999686

...

187(0.58155, 0.30900, 0.10945); maxRE: 0.06266263268226184

188(0.58155, 0.30900, 0.10945); maxRE: 0.06266258786776895

189(0.58155, 0.30900, 0.10945); maxRE: 0.06266253816672833

...

191(0.58155, 0.30900, 0.10945); maxRE: 0.06266229703314807

...

197(0.58155, 0.30900, 0.10945); maxRE: 0.06266224017021971

198(0.58155, 0.30900, 0.10945); maxRE: 0.06266216603176003

...

215(0.58155, 0.30900, 0.10945); maxRE: 0.06266214523195712

...

220(0.58155, 0.30900, 0.10945); maxRE: 0.06266212454962528

...

234(0.58155, 0.30900, 0.10945); maxRE: 0.06266212245396349

...

244(0.58155, 0.30900, 0.10945); maxRE: 0.06266211273247646

...

264(0.58155, 0.30900, 0.10945); maxRE: 0.0626621122171775

...

271(0.58155, 0.30900, 0.10945); maxRE: 0.06266211174435549

...

292(0.58155, 0.30900, 0.10945); maxRE: 0.06266211162918314

...

303(0.58155, 0.30900, 0.10945); maxRE: 0.06266211151923393

...

307(0.58155, 0.30900, 0.10945); maxRE: 0.06266211148551928

...

315(0.58155, 0.30900, 0.10945); maxRE: 0.06266211145915544

...

317(0.58155, 0.30900, 0.10945); maxRE: 0.06266211144582226

...

337(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140852077

...

366(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140680279

367(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140623873

...

370(0.58155, 0.30900, 0.10945); maxRE: 0.0626621114057638

...

373(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140280156

...

377(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140272415

...

383(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140200457

...

397(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140170644

...

415(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140170323

...

419(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140160553

...

427(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140155047

428(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140154603

...

436(0.58155, 0.30900, 0.10945); maxRE: 0.0626621114014693

...

455(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140146591

...

463(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140146494

...

465(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140146219

...

467(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140146184

...

483(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140145761

...

489(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140145735

...

492(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140145672

...

526(0.58155, 0.30900, 0.10945); maxRE: 0.06266211140145668

...

530(0.58155, 0.30900, 0.10945); maxRE: 0.0626621114014565

...

Result after 1000 iteration:

(0.58155, 0.30900, 0.10945)

From the above result, it is not difficult to find out that the final result of maxRE is the same as the initial (up to 5 digits after the decimal place). The other two have the same solution which is relatively different from the initial solution. So, intuition tells that there may be a "bug" in DE optimization and trying R package to validate this assumption seems a good idea.

10.7 'DEoptim' Package in R

10.7.1 Description

Performs evolutionary global optimization via the Differential Evolution algorithm.

10.7.2 Usage

DEoptim(fn, lower, upper, control = DEoptim.control(), ..., fnMap=NULL)

10.7.3 Arguments

- 1. fn: the function to be optimized (minimized). The function should have as its first argument the vector of real-valued parameters to optimize, and return a scalar real result. NA and NaN values are not allowed.

- 2. lower, upper: two vectors specifying scalar real lower and upper bounds on each parameter to be optimized, so that the i-th element of lower and upper applies to the i-th parameter. The implementation searches between lower and upper for the global optimum (minimum) of fn.

- 3. control: a list of control parameters; see DEoptim.control.

- 4. fnMap: an optional function that will be run after each population is created, but before the population is passed to the objective function. This allows the user to impose integer/cardinality constriants.

- 5. ...: further arguments to be passed to fn.

10.7.4 Details About 'DEoptim' Package

The R implementation of Differential Evolution (DE), DEoptim, was first published on the Comprehensive R Archive Network (CRAN) in 2005 by David Ardia. Early versions were written in pure R. Since version 2.0-0 (published to CRAN in 2009) the package has relied on an interface to a

C implementation of DE, which is significantly faster on most problems [3]. Since version 2.0-3 the C implementation dynamically allocates the memory required to store the population, removing limitations on the number of members in the population and length of the parameter vectors that may be optimized. Since version 2.2-0, the package allows for parallel operation, so that the evaluations of the objective function may be performed using all available cores. This is accomplished using either the built-in parallel package or the foreach package. If the parallel operation is desired, the user should set parallelType and make sure that the arguments and packages needed by the objective function are available; see DEoptim.control, the example below and examples in the sandbox directory for details [33]. This simplistic strategy usually does not work all that well for gradient-based or Newton-type methods. It is likely to be all right when the solution is in the interior of the feasible region, but when the solution is on the boundary, the optimization algorithm would have a difficult time converging. Furthermore, when the solution is on the boundary, this strategy would make the algorithm converge to an inferior solution in the interior. However, for methods such as DE which are not gradient based, this strategy might not be that bad [33]. DEoptim relies on the repeated evaluation of the objective function in order to move the population toward a global minimum. Users interested in making DEoptim run as fast as possible should consider using the package in parallel mode (so that all CPU's available are used), and also ensure that evaluation of the objective function is as efficient as possible (e.g. by using

vectorization in pure R code, or writing parts of the objective function in a lower-level language like C or Fortran) [33].

10.7.5 Values

The output of the function DEoptim is a member of the S3 class DEoptim. More precisely, this is a list (of length 2) containing the following elements [2]:

1. optim, a list containing the following elements:

   - bestmem: the best set of parameters found.
   - bestval: the value of fn corresponding to bestmem.
   - nfeval: number of function evaluations.
   - iter: number of procedure iterations.

2. member, a list containing the following elements:

   - lower: the lower boundary.
   - upper: the upper boundary.
   - bestvalit: the best value of fn at each iteration.
   - bestmemit: the best member at each iteration.
   - pop: the population generated at the last iteration.
   - storepop: a list containing the intermediate populations.

10.7.6 rsrDE Custom Function

Using maxRE as evaluation function to optimaze PC matrices (n = 3).

```r
# using maxRE as evaluation function to optimaze PC matrices (x = 3).
rsrDE <- function(initial_vector)
{
    nsi_3 <- function(x)
    {
        w_initial = x
        re_pc = c(
            1,
            w_initial[1] / w_initial[2],
            w_initial[1] / w_initial[3],
            1 / (w_initial[1] / w_initial[2]),
            1,
            w_initial[2] / w_initial[3],
            1 / (w_initial[1] / w_initial[3]),
            1 / (w_initial[2] / w_initial[3]),
            1
        )
        reconstruct_PC = matrix(re_pc, nrow = 3, byrow = TRUE)
        result = c(0, 0, 0, 0, 0, 0, 0, 0, 0)
        # The maximum relative error between the newly generated PC matrix
        # and the original given PC matrix is obtained here.
        count = 1
        for (i in 1:3)
        {
            for (j in 1:3)
            {
                result[count] = abs(w_initial_pc_matrix[i, j] - reconstruct_PC[i, j]) /
                    w_initial_pc_matrix[i, j]
                count = count + 1
            }
            count = count + 1
        }
        max(result)
    }
```

Figure 31: rsrDE Function

In figure 31, nsi_3 is the function to be optimized (minimized). The function should have as its first argument the vector of real-valued parameters to optimize and return a real scalar result. x is the vector after several iterations; it should be a little bit different from the initial vector. The test Code for experiments with rsrDE is shown in figure 32 below:

```r
# test
matrix_value <-
    c(1.00000,
      2.00000,
      5.00000,
      0.50000,
      1.00000,
      3.00000,
      0.20000,
      0.33333,
      1.00000)
nsi_matrix <- matrix(matrix_value, nrow = 3, byrow = T)
#      [,1]    [,2] [,3]
# [1,]  1.0 2.00000    5
# [2,]  0.5 1.00000    3
# [3,]  0.2 0.33333    1
solution_ev(nsi_matrix)
# [1] 0.5815523 0.3089957 0.1094520
rsrDE(solution_ev(nsi_matrix))
```

Figure 32: Test rsrDE Function

The test results for case 11:

The initial vector(EV) is: (0.5815523 0.3089957 0.1094520);

Iteration: 1 bestvalit: 0.001839 bestmemit: 0.579890 0.308679 0.109301

Iteration: 2 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 3 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 4 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 5 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 6 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 7 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 8 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 9 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 10 bestvalit: 0.000305 bestmemit: 0.586371 0.311617 0.110347

Iteration: 11 bestvalit: 0.000284 bestmemit: 0.583681 0.310075 0.109866

Iteration: 12 bestvalit: 0.000284 bestmemit: 0.583681 0.310075 0.109866

Iteration: 13 bestvalit: 0.000284 bestmemit: 0.583681 0.310075 0.109866

Iteration: 14 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 15 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 16 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 17 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 18 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 19 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 20 bestvalit: 0.000179 bestmemit: 0.585817 0.311232 0.110235

Iteration: 21 bestvalit: 0.000097 bestmemit: 0.585918 0.311285 0.110268

Iteration: 22 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 23 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 24 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 25 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 26 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 27 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 28 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 29 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 30 bestvalit: 0.000056 bestmemit: 0.585732 0.311201 0.110232

Iteration: 31 bestvalit: 0.000033 bestmemit: 0.585397 0.311047 0.110175

Iteration: 32 bestvalit: 0.000033 bestmemit: 0.585397 0.311047 0.110175

Iteration: 33 bestvalit: 0.000033 bestmemit: 0.585397 0.311047 0.110175

Iteration: 34 bestvalit: 0.000033 bestmemit: 0.585397 0.311047 0.110175

Iteration: 35 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 36 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 37 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 38 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 39 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 40 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 41 bestvalit: 0.000029 bestmemit: 0.584587 0.310607 0.110020

Iteration: 42 bestvalit: 0.000025 bestmemit: 0.583931 0.310267 0.109901

Iteration: 43 bestvalit: 0.000016 bestmemit: 0.585347 0.311007 0.110165

Iteration: 44 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 45 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 46 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 47 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 48 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 49 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 50 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 51 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 52 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 53 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 54 bestvalit: 0.000015 bestmemit: 0.585460 0.311077 0.110188

Iteration: 55 bestvalit: 0.000010 bestmemit: 0.583698 0.310134 0.109855

Iteration: 56 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 57 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 58 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 59 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 60 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 61 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 62 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 63 bestvalit: 0.000006 bestmemit: 0.584886 0.310767 0.110079

Iteration: 64 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 65 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 66 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 67 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 68 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 69 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 70 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 71 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 72 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 73 bestvalit: 0.000003 bestmemit: 0.585150 0.310908 0.110129

Iteration: 74 bestvalit: 0.000003 bestmemit: 0.584543 0.310584 0.110015

Iteration: 75 bestvalit: 0.000003 bestmemit: 0.584543 0.310584 0.110015

Iteration: 76 bestvalit: 0.000003 bestmemit: 0.584543 0.310584 0.110015

Iteration: 77 bestvalit: 0.000003 bestmemit: 0.584543 0.310584 0.110015

Iteration: 78 bestvalit: 0.000002 bestmemit: 0.584702 0.310669 0.110045

Iteration: 79 bestvalit: 0.000002 bestmemit: 0.585738 0.311220 0.110240

Iteration: 80 bestvalit: 0.000002 bestmemit: 0.585738 0.311220 0.110240

Iteration: 81 bestvalit: 0.000001 bestmemit: 0.584286 0.310448 0.109966

Iteration: 82 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 83 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 84 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 85 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 86 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 87 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 88 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 89 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 90 bestvalit: 0.000001 bestmemit: 0.585213 0.310941 0.110141

Iteration: 91 bestvalit: 0.000001 bestmemit: 0.586000 0.311359 0.110289

Iteration: 92 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 93 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 94 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 95 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 96 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 97 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 98 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 99 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Iteration: 100 bestvalit: 0.000000 bestmemit: 0.583972 0.310281 0.109907

Result: $maxre$ = 3.387878e-07 (WRONG!)

10.7.7 Analysis of Parallel Experiments between Java and R

It is not difficult to find that the results converge from 92th generation. The maximum relative error for R calculated is 0.0000004492711. The results implemented in Java code converge from 530th generation. The maximum relative error is 0.0626621114014565. It is concluded that we may need to modify some parts of Java code to make the maxRE value in the result smaller. Putting the results from R experiments together with those from Java experiments (Generation 100th) below to facilitate comparison and checking:

	A	B	C
1	**Initial PC matrix:**		
2	1. 0000000	2. 0000000	5. 0000000
3	0. 5000000	1. 0000000	3. 0000000
4	0. 2000000	0. 3333333	1. 0000000
5			
6	**Case1: Evolved PC Matrix By Java:**		
7	1. 0000000	1. 8817546	5. 3111354
8	0. 5314189	1. 0000000	2. 8224379
9	0. 1882837	0. 3543036	1. 0000000
10	maxRE: 0.06292154164312601		
11	Convergence Location: 530th		
12			
13	**Case2: Evolved PC Matrix By R:**		
14	1. 0000000	1. 8820717	5. 3133090
15	0. 5313294	1. 0000000	2. 8231180
16	0. 1882066	0. 3542183	1. 0000000
17	maxRE: 3.387878e-07		
18	Convergence Location: 92th		

Figure 33: Results of Parallel Experiment

Using a testing function to check the maxRE got in case 2. Figure 34 shows the test code I created and the running result:

```
public class TestMaxRe {
    public static void main(String[] args) {
        double[][] M = {{1.0000000, 2.0000000, 5.000000}, {0.5000000, 1.0000000, 3.000000},
{0.2000000, 0.3333333, 1.000000}};
        double[][] A = {{1.0000000, 1.8820717, 5.313309}, {0.5313294, 1.0000000, 2.823118},
{0.1882066, 0.3542183, 1.000000}};
        double maxRE = EvaluationFunction.maxRE(M, A);
        System.out.println("maxRE:" + maxRE);
    }
}
```

```
maxRE:0.06266180000000006
```

Figure 34: TestMaxRe() Class

From the result section of the figure above, we can see some problems: the evolved PC matrices are quite close after the 100th generation. However, the value of maxRE is not the same as that obtained in Case 2 which is 3.387878e-07.

E	F	G	H
Java	**REcol_1**	**REcol_2**	**REcol_3**
RErow_1	0. 0000000	0. 0591227	0. 0622271
RErow_2	0. 0628378	0. 0000000	0. 0591874
RErow_3	0. 0585815	0. 0629109	0. 0000000
R	**REcol_1**	**REcol_2**	**REcol_3**
RErow_1	0. 0000000	0. 0589642	0. 0626618
RErow_2	0. 0626588	0. 0000000	0. 0589607
RErow_3	0. 0589670	0. 0626550	0. 0000000

Figure 35: Use Excel to Calculate the maxRE

As shown in the excel screenshot above, the relative error values for each position of the PC matrix are calculated and compared the results of Java and R. The correctness of the results calculated by the test program in figure 34 can be further verified. That is to say, the value of maxRE obtained by R experimental program is wrong.

It is necessary to compare the results of R experiment with those of Java experiment. Because the differential evolution algorithm is a heuristic genetic algorithm, we expect the ideal result of comparison: the results of R experiment are very similar but not entirely the same. Specifically, the vectors obtained by each generation of the algorithm are "sufficiently similar but not identical."

After iterating over 1000 generations of the original PC matrix with Java experimental code, I observed that the convergence position of this algorithm was 488 generations. However, if only six decimal places are taken into account (in order to keep the accuracy of R experimental results, i.e., control variables), the convergence position of the algorithm is 161 generations, at which time the value of maxRE is 0.062662. Similarly, for the experimental results of the Java version code, we compare the results of the 100 generation code (62th-161th) with those of the R version code. The experimental results are summarized as shown in the following figure next page. The supervisor suggests that I should use Euclidean distance as a measure function. The experimental results are summarized as shown in the following figure.

I2 =SQRT((B2-F2)^2+(C2-G2)^2+(D2-H2)^2)

	A	B	C	D	E	F	G	H	I
1	r	r_vec_pos01	r_vec_pos02	r_vec_pos03	java	j_vec_pos01	j_vec_pos02	j_vec_pos03	ED beteween r and java
2	Gen001	0.579890	0.308679	0.109301	Gen062	0.581544	0.308997	0.109459	0.0016917
3	Gen002	0.586371	0.311617	0.110347	Gen063	0.581544	0.308997	0.109459	0.0055635
4	Gen003	0.586371	0.311617	0.110347	Gen064	0.581544	0.308997	0.109459	0.0055635
5	Gen004	0.586371	0.311617	0.110347	Gen065	0.581544	0.308997	0.109459	0.0055635
6	Gen005	0.586371	0.311617	0.110347	Gen066	0.581544	0.308997	0.109459	0.0055635
7	Gen006	0.586371	0.311617	0.110347	Gen067	0.581544	0.308997	0.109459	0.0055635
8	Gen007	0.586371	0.311617	0.110347	Gen068	0.581544	0.308997	0.109459	0.0055635
9	Gen008	0.586371	0.311617	0.110347	Gen069	0.581544	0.308997	0.109459	0.0055635
10	Gen009	0.586371	0.311617	0.110347	Gen070	0.581544	0.308997	0.109459	0.0055635
11	Gen010	0.586371	0.311617	0.110347	Gen071	0.581544	0.308997	0.109459	0.0055635
12	Gen011	0.583681	0.310075	0.109866	Gen072	0.581544	0.308997	0.109459	0.0024279
13	Gen012	0.583681	0.310075	0.109866	Gen073	0.581544	0.308997	0.109459	0.0024279
14	Gen013	0.583681	0.310075	0.109866	Gen074	0.581544	0.308997	0.109459	0.0024279
15	Gen014	0.585817	0.311232	0.110235	Gen075	0.581544	0.308997	0.109459	0.0048843
16	Gen015	0.585817	0.311232	0.110235	Gen076	0.581544	0.308997	0.109459	0.0048843
17	Gen016	0.585817	0.311232	0.110235	Gen077	0.581544	0.308997	0.109459	0.0048843
18	Gen017	0.585817	0.311232	0.110235	Gen078	0.581544	0.308997	0.109459	0.0048843
19	Gen018	0.585817	0.311232	0.110235	Gen079	0.581544	0.308997	0.109459	0.0048843
20	Gen019	0.585817	0.311232	0.110235	Gen080	0.581548	0.309005	0.109447	0.0048790
21	Gen020	0.585817	0.311232	0.110235	Gen081	0.581548	0.309005	0.109447	0.0048790
22	Gen021	0.585918	0.311285	0.110268	Gen082	0.581548	0.309005	0.109447	0.0049969

Figure 36: Euclidean Distance between R and Java (1)

I101 =SQRT((B101-F101)^2+(C101-G101)^2+(D101-H101)^2)

	A	B	C	D	E	F	G	H	I
80	Gen079	0.585738	0.311220	0.110240	Gen140	0.581552	0.308996	0.109452	0.0048052
81	Gen080	0.585738	0.311220	0.110240	Gen141	0.581552	0.308996	0.109452	0.0048052
82	Gen081	0.584286	0.310448	0.109966	Gen142	0.581552	0.308996	0.109452	0.0031380
83	Gen082	0.585213	0.310941	0.110141	Gen143	0.581552	0.308996	0.109452	0.0042025
84	Gen083	0.585213	0.310941	0.110141	Gen144	0.581552	0.308996	0.109452	0.0042025
85	Gen084	0.585213	0.310941	0.110141	Gen145	0.581552	0.308996	0.109452	0.0042025
86	Gen085	0.585213	0.310941	0.110141	Gen146	0.581552	0.308996	0.109452	0.0042025
87	Gen086	0.585213	0.310941	0.110141	Gen147	0.581552	0.308996	0.109452	0.0042025
88	Gen087	0.585213	0.310941	0.110141	Gen148	0.581552	0.308996	0.109452	0.0042025
89	Gen088	0.585213	0.310941	0.110141	Gen149	0.581552	0.308996	0.109452	0.0042025
90	Gen089	0.585213	0.310941	0.110141	Gen150	0.581552	0.308996	0.109452	0.0042025
91	Gen090	0.585213	0.310941	0.110141	Gen151	0.581552	0.308996	0.109452	0.0042025
92	Gen091	0.586000	0.311359	0.110289	Gen152	0.581552	0.308996	0.109452	0.0051058
93	Gen092	0.583972	0.310281	0.109907	Gen153	0.581552	0.308996	0.109452	0.0027775
94	Gen093	0.583972	0.310281	0.109907	Gen154	0.581552	0.308996	0.109452	0.0027775
95	Gen094	0.583972	0.310281	0.109907	Gen155	0.581552	0.308996	0.109452	0.0027775
96	Gen095	0.583972	0.310281	0.109907	Gen156	0.581552	0.308996	0.109452	0.0027775
97	Gen096	0.583972	0.310281	0.109907	Gen157	0.581552	0.308996	0.109452	0.0027775
98	Gen097	0.583972	0.310281	0.109907	Gen158	0.581552	0.308996	0.109452	0.0027775
99	Gen098	0.583972	0.310281	0.109907	Gen159	0.581552	0.308996	0.109452	0.0027775
100	Gen099	0.583972	0.310281	0.109907	Gen160	0.581552	0.308996	0.109452	0.0027775
101	Gen100	0.583972	0.310281	0.109907	Gen161	0.581552	0.308997	0.109452	0.0027771

Figure 37: Euclidean Distance between R and Java (2)

The Euclidean distance between R and Java is 0.0027771 at the final convergence position (sixth place after the decimal point).

10.7.8 Correction of Errors in R

Next, we need to correct the errors in the R experimental code. That is to say; the rsrDE part is modified to make the maxRE value fall within the normal range.

Firstly, I tracked the value of the local variable "result", and found that there were problems in calculating the nine relative errors stored (five orders of magnitude different from the correct value). Then I tracked local variables w_initia_pc_matrix and reconstruct_PC. Elements in w_initial_pc_matrix and reconstruct_PC:

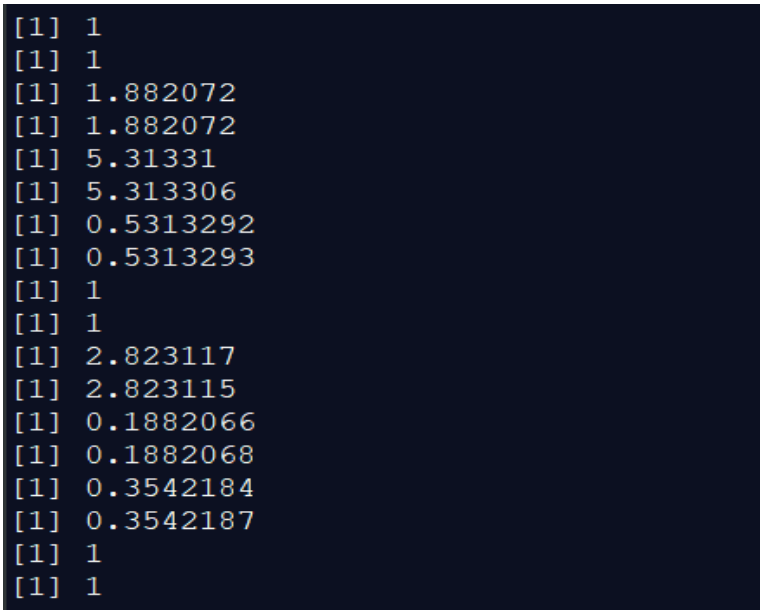


Figure 38: Elements before Correction

As shown in figure 38 above, it was found that some given values of the original PC matrix, such as 2.000000, 3.000000 and 5.000000, did not exist

in the 18 elements of the two matrices printed by the program. This means that the problem lies in the original matrix w_initial_pc_matrix. Through the modification of w_initial_pc_matrix, we got the correct result of $maxRE$. The modification part is shown in figure 39:

```
w_initial = initial_vector
w_initial_pc = c(
    1,
    w_initial[1] / w_initial[2],
    w_initial[1] / w_initial[3],
    1 / (w_initial[1] / w_initial[2]),
    1,
    w_initial[2] / w_initial[3],
    1 / (w_initial[1] / w_initial[3]),
    1 / (w_initial[2] / w_initial[3]),
    1
)
```

Figure 39: Error Code in R

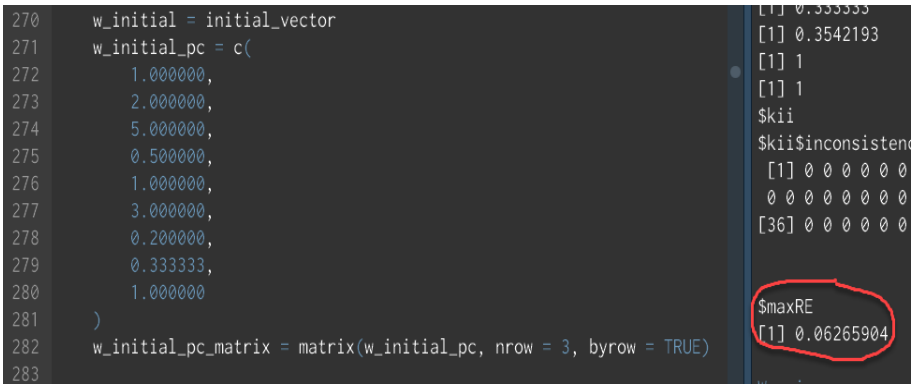


Figure 40: Correction and New Running Result

The two screenshots above show the status of the error code before and after modification, respectively. The red pen circled at the bottom right of

Figure 40 is the correct result, and the new running result are shown as Figure 41 below:

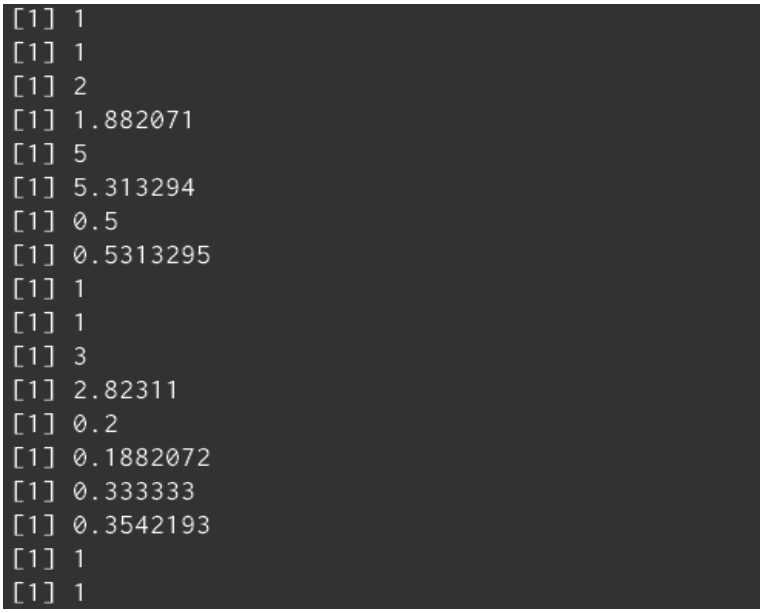


Figure 41: Elements after Correction

Looking at Figure 41, we can easily see that the element values in the original PC matrix appear, which shows that our modification of the corresponding error code part of the R code is correct.

10.7.9 Generation of NSI PC Matrix

According to the requirement of the experiment, we need to analyze and summarize some characteristics of NSI PC matrix from a statistical point of view. For each dimension of the NSI PC matrix (n = 3, 4,..., 8), 100 000 random samples need to be generated separately. So we need a total of

600,000 randomly generated NSI PC matrices. In these matrices, the values of all elements are between 1/3 and 3 (including boundary values). With the help of the "java.Random" class and IO stream technology in Java, we can quickly generate random NSI PC matrices that meet the requirements and store the generated matrices.

```java
public static double[][] deviateFullyConsistentPCM(double[][] fullyConsistentPCM) {
    // the deviation factor
    double fluctuationFactor;
    // upper limit to create random numbers
    int upperLimit = 20000;
    // lower limit to create random numbers
    int lowerLimit = 0;
    // PC matrix after deviate operation
    double[][] deviatedPCMatrix = new double[fullyConsistentPCM.length][fullyConsistentPCM.length];
    // temporary array
    double[][] tempMatrix = new double[fullyConsistentPCM.length][fullyConsistentPCM.length];

    // traverse to get every element in a PC matrix
    for (int i = 0; i < deviatedPCMatrix.length; i++) {
        for (int j = 0; j < deviatedPCMatrix[i].length; j++) {
            // the object of deviation is each element above the principal diagonal line of the PC matri
            if (i < j) {
                // the magnitude of fluctuation(deviation) is between -20% to +20%
                Random random = new Random();
                double temp = (random.nextInt( bound: upperLimit - lowerLimit) + lowerLimit) / 100000.0;
                // about half of the target elements increase their values by 20%
                if ((i + j) % 2 == 0) {
                    fluctuationFactor = (double) Math.round((1 + temp) * 100000) / 100000;
                } else {
                    // about half of the target elements decrease their values by 20%
                    fluctuationFactor = (double) Math.round((1 - temp) * 100000) / 100000;
                }
                // a group of fluctuationFactor values when deviate by 20% and x = 4:
                // 0.92449, 1.08640, 0.83382, 0.81936, 1.18805, 0.84441
                // System.out.println("fluctuationFactor: " + fluctuationFactor);

                deviatedPCMatrix[i][j] = fullyConsistentPCM[i][j] * fluctuationFactor;
                tempMatrix[i][j] = deviatedPCMatrix[i][j];
            } else if (i == j) {
                deviatedPCMatrix[i][j] = 1.00000;
            } else {
                /*In order to maintain the relative stability of the whole PC matrix system,
                we need to transform the elements below the principal diagonal appropriately*/
                deviatedPCMatrix[i][j] = 1.00000 / tempMatrix[j][i];
            }
            System.out.println("deviatedPCMatrix[i][j]: " + deviatedPCMatrix[i][j]);
        }
        // System.out.println("----------------------------------------------------");
    }
    // return a deviated PC matrix
    return deviatedPCMatrix;
}
```

Figure 42: deviateFullyConsistentPCM() Function

Traversing to get every element in an entirely consistent PC matrix, the object of deviation is each element above the main diagonal line of the PC matrix. The magnitude of fluctuation (deviation) is between -20% to +20%, about half of the target elements increase their values by 20% and nearly half of the target elements decrease their values by 20%. In order to maintain the relative stability of the whole PC matrix system, we need to transform the elements below the principal diagonal appropriately. Besides, we need a filter. As shown in figure 43, all values of the element in an NSI PC matrix created should between minElement (0.33333) to maxElement (3.00000).

```
public static double[][] filterMatrix(double[][] randomPCMatrix) {
    double maxElement = 3.00000;
    double minElement = 0.33333;

    for (int x = 0; x < randomPCMatrix.length; x++) {
        for (int y = 0; y < randomPCMatrix[x].length; y++) {
            if (randomPCMatrix[x][y] > maxElement) {
                randomPCMatrix[x][y] = maxElement;
            } else if (randomPCMatrix[x][y] < minElement) {
                randomPCMatrix[x][y] = minElement;
            } else {
                randomPCMatrix[x][y] = randomPCMatrix[x][y];
            }
        }
    }
    return randomPCMatrix;
}
```

Figure 43: filterMatrix() Function

11 Summary

In this section, both histograms and box plots will be used for data visualization and data analysis. The most significant advantage of box diagram is that it is not affected by outliers, can accurately and steadily depict the discrete distribution of data, but also conducive to data cleaning. The frequency distribution histogram can clearly show the frequency distribution of each group, and it is easy to show the frequency difference between groups. Besides, describe() function in R language can be adopted to extract the values of some commonly used statistical variables quickly and effectively.

11.1 Box Plot

Box plot is a statistical chart used to display a set of data dispersion. It is often used in various fields, often in quality management. It is mainly used to reflect the distribution characteristics of the original data, and can also compare the distribution characteristics of multiple groups of data.

Through observation (Figure 44 and Figure 45 next page), we can preliminarily conclude that outlier is the value of maxRE greater than 0.10 in the sample and most of the valid data sets are between 0.4 and 0.6.

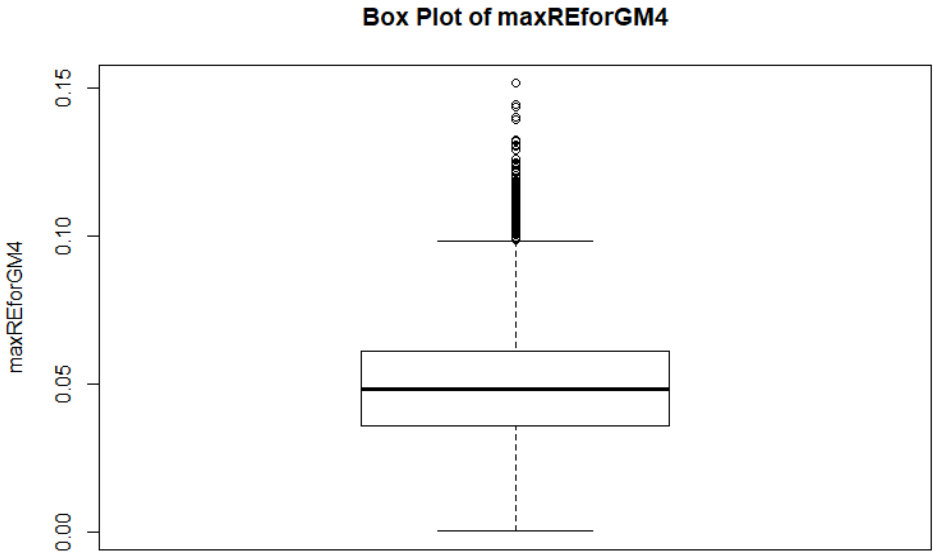


Figure 44: Box Plot of maxREforGM (n=4)

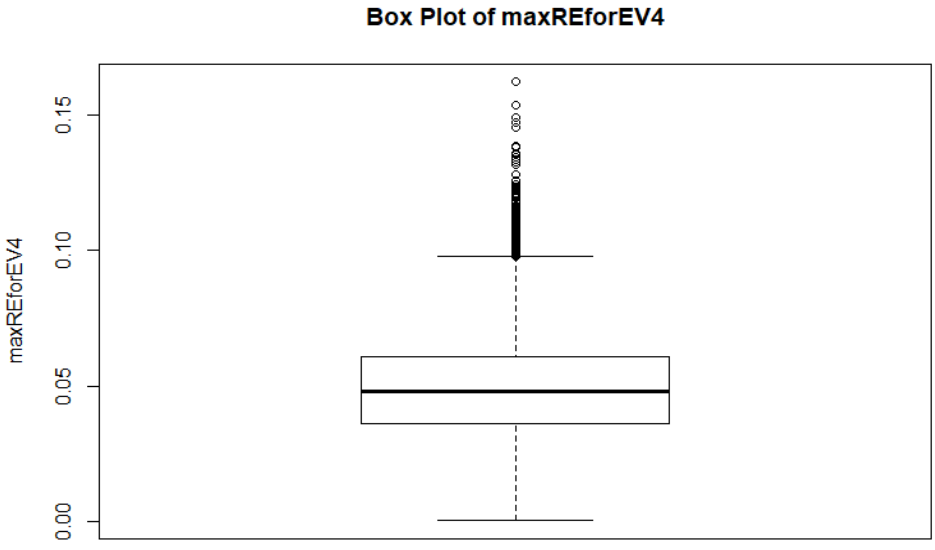


Figure 45: Box Plot of maxREforEV (n=4)

11.2 Histogram

Histogram, the horizontal axis represents the data type, and the vertical axis represents the distribution.

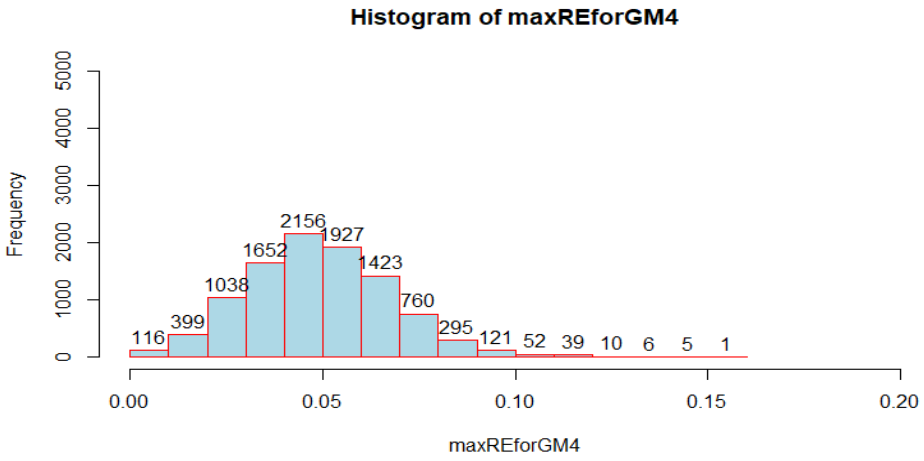


Figure 46: Histogram of maxREforGM (n=4)

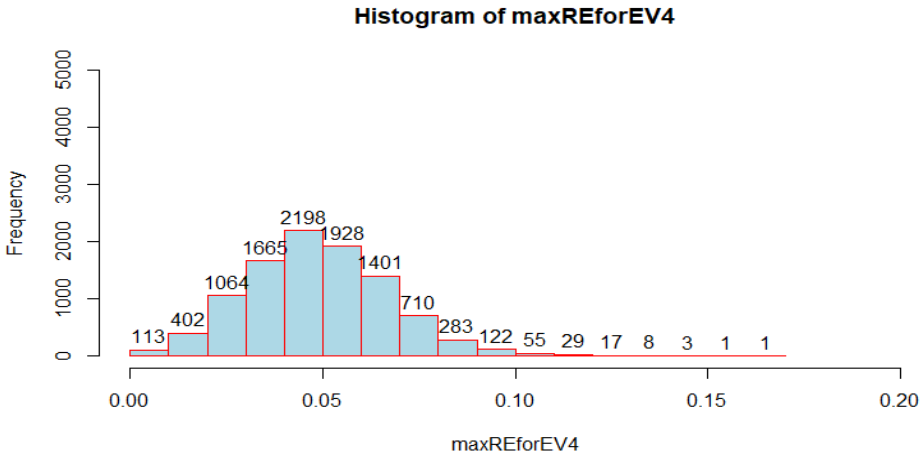


Figure 47: Histogram of maxREforEV (n=4)

Through the analysis and comparison of Figure 46 and Figure 47, we can see that the values of maxRE are widely distributed between 0 and 0.1, whether using GM or EV as the initial vector. The value of maxRE shows a positive skewness distribution trend: the frequency near 0.5 is the highest, the closer to the boundary (0 or 0.1), the lower the frequency. If we use the description() function in the psych package of the R language, we can get more precise values of some statistical variables, such as sample mean, variance, standard deviation, kurtosis and so on. As shown in Figure 48 below:

n	mean	sd	median	trimmed	mad	min	max	range	skew	kurtosis	se
10000	0. 0492354	0. 0190213	0. 048387	0. 0486413	0. 0185489	0. 0002769	0. 1517149	0. 151438	0. 4673159	0. 8461659	0. 0001902
10000	0. 0489487	0. 0189797	0. 0479899	0. 0483127	0. 0182951	0. 0002811	0. 1622536	0. 1619725	0. 5127378	1. 06443	0. 0001898

Figure 48: Statistical Parameters (n=4)

Generally speaking, the value of maxRE with GM as the initial parameter of the algorithm is slightly smaller than that with EV as the initial parameter of the algorithm. It shows that EV as the initial parameter of the algorithm can achieve a better evolutionary effect. From the standard deviation, we can see that the effect of the DE algorithm is more stable when EV is the initial parameter of the algorithm. From the skewness value, we can see that the value of maxRE obtained with GM as the initial parameter of the algorithm is more satisfying than that obtained with EV as the initial parameter of the algorithm.

11.3 Summary of Monte Carlo Experiment

n	d	maxKii	pev	maxRE		trunR	evWins
				GM	EV		
5	0.1	0.1720330	5.0060990	0.0218295	0.0218447	3.2%	49.5%
5	0.2	0.3202239	5.0251340	0.0458103	0.0457429	9.7%	52.0%
5	0.3	0.4438558	5.0578270	0.0736312	0.0732445	26.3%	54.4%
5	0.4	0.5407155	5.1054250	0.1010554	0.1007876	55.5%	51.9%
5	0.5	0.6193592	5.1647360	0.1359702	0.1339794	123.6%	56.1%
5	0.6	0.6789396	5.2380080	0.1764090	0.1716217	261.6%	56.3%
5	0.7	0.7026682	5.2718030	0.1877054	0.1828145	681.7%	53.6%
5	0.8	0.7104865	5.2901670	0.1939778	0.1877525	1609.4%	57.4%
6	0.1	0.1742612	6.0073260	0.0301662	0.0301595	19.7%	49.9%
6	0.2	0.3211983	6.0299400	0.0627438	0.0627709	48.5%	50.5%
6	0.3	0.4439703	6.0680770	0.0971337	0.0964119	89.3%	52.0%
6	0.4	0.5465872	6.1256130	0.1369691	0.1352923	154.8%	56.6%
6	0.5	0.6323225	6.2011240	0.1874986	0.1836822	315.5%	59.0%
6	0.6	0.7024754	6.2922830	0.2477434	0.2383516	766.6%	60.0%
6	0.7	0.7207773	6.3397290	0.2655126	0.2573131	2646.4%	56.9%
6	0.8	0.7302408	6.3642600	0.2666606	0.2577188	8935.0%	57.1%
7	0.1	0.1894896	7.0093520	0.0284464	0.0284401	4.3%	48.6%
7	0.2	0.3458936	7.0384070	0.0600849	0.0600051	19.9%	50.4%
7	0.3	0.4772013	7.0880840	0.0969838	0.0970575	56.6%	50.0%
7	0.4	0.5840607	7.1612360	0.1396295	0.1390744	125.6%	51.4%
7	0.5	0.6664400	7.2596960	0.1979396	0.1946920	304.3%	55.7%
7	0.6	0.7268868	7.3665750	0.2549888	0.2518094	987.7%	55.1%
7	0.7	0.7518546	7.4307090	0.2828553	0.2760326	4740.8%	56.7%
7	0.8	0.7642377	7.4660770	0.2769351	0.2743985	24896.5%	54.5%
8	0.1	0.1881716	8.0106480	0.0330586	0.0330067	28.1%	53.3%
8	0.2	0.3469607	8.0440270	0.0700391	0.0700541	68.1%	49.1%
8	0.3	0.4799036	8.1018620	0.1110569	0.1103802	147.0%	53.0%
8	0.4	0.5888724	8.1848090	0.1609495	0.1602257	311.5%	54.6%
8	0.5	0.6817616	8.3016940	0.2258288	0.2221284	787.5%	55.6%
8	0.6	0.7408235	8.4296470	0.2953621	0.2881176	2871.6%	57.3%
8	0.7	0.7647386	8.5019370	0.3303131	0.3234346	22234.9%	57.2%
8	0.8	0.7714507	8.5427080	0.3213509	0.3165324	192689.4%	54.9%

Figure 49: Results of Monte Carlo Experiment

Using Java code, we randomly generated 10,000 NSI PC matrices satisfying certain requirements, and recorded some important results of the experiment in the table shown in figure 49. In figure 49, n denotes the dimension of the NSI PC matrix; D denotes the fluctuation amplitude imposed on the target elements in the matrix; maxKii denotes the mean value of the maximum Kii of the matrix; and PEV denotes the mean value of the principal eigenvalue of the matrix. maxRE (GM) and maxRE (EV) denote the mean value of the maximum relative error between the evolved matrix and the original matrix (GM and EV); trunR denotes the truncate rate in the process of generating the matrices; evWins denotes that the frequency of maxRE (EV) is less than that of maxRE (GM).

From figure 49, we can see that the value of maxKii increases with the increase of matrix dimension. It shows that the larger the matrix dimension is, the higher the inconsistency is when the same fluctuation amplitude is applied. It is not difficult to understand that the value of trunR increases significantly with the increase of matrix dimension and fluctuation amplitude. In addition, on the whole, EV as the initial input of DE algorithm can achieve better evolutionary results than GM as the initial input of DE algorithm.

References

[1] J Allaire. Rstudio: integrated development environment for r. *Boston, MA*, 2012.

[2] David Ardia, Kris Boudt, Peter Carl, Katharine Mullen, and Brian G Peterson. Differential evolution with deoptim: an application to non-convex portfolio optimization. *The R Journal*, 3(1):27–34, 2011.

[3] David Ardia, Katharine M Mullen, Joshua Ulrich, and Brian G Peterson. Deoptim: An r package for global optimization by differential evolution. *Journal of Statistical Software*, 40(6):1–26, 2011.

[4] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[5] Jonathan Barzilai. Deriving weights from pairwise comparison matrices. *Journal of the operational research society*, 48(12):1226–1232, 1997.

[6] Richard Bellman. *Introduction to matrix analysis*, volume 19. Siam, 1997.

[7] Joseph Bergin, Mark Stehlik, Jim Roberts, and Richard Pattis. *Karel J. Robot: A gentle introduction to the art of object-oriented programming in Java*. Dream Songs Redwood City, 2005.

[8] Matts Björck. Fitting with differential evolution: an introduction and evaluation. *Journal of Applied Crystallography*, 44(6):1198–1204, 2011.

[9] Joshua Bloch. *Effective java (the java series)*. Prentice Hall PTR, 2008.

[10] Heiko Böck. Intellij idea and the netbeans platform. In *The Definitive Guide to NetBeans^{TM} Platform 7*, pages 431–437. Springer, 2012.

[11] Sándor Bozóki and Tamás Rapcsák. On saaty's and koczkodaj's inconsistencies of pairwise comparison matrices. *Journal of Global Optimization*, 42(2):157–175, 2008.

[12] Matteo Brunelli. Studying a set of properties of inconsistency indices for pairwise comparisons. *Annals of Operations Research*, 248(1-2):143–161, 2017.

[13] Mary Campione and Kathy Walrath. *The Java Tutorial: Object-Oriented Programming for the Internet (Book/CD)*. Addison-Wesley Longman Publishing Co., Inc., 1998.

[14] John Chambers. *Software for data analysis: programming with R*. Springer Science & Business Media, 2008.

[15] Alexandra M Freund and Paul B Baltes. Selection, optimization, and compensation as strategies of life management: correlations with subjective indicators of successful aging. *Psychology and aging*, 13(4):531, 1998.

[16] Jándor Fülöp, Waldemar W Koczkodaj, and Stanisław J Szarek. A different perspective on a scale for pairwise comparisons. In *Transactions on computational collective intelligence I*, pages 71–84. Springer, 2010.

[17] Mitsuo Gen and Lin Lin. Genetic algorithms. *Wiley Encyclopedia of Computer Science and Engineering*, pages 1–15, 2007.

[18] Michael W Herman and Waldemar W Koczkodaj. A monte carlo study of parwise comparison. *Inf. Process. Lett.*, 57(1):25–29, 1996.

[19] Joe Hicklin, Cleve Moler, Peter Webb, Ronald F Boisvert, Bruce Miller, Roldan Pozo, and Karin Remington. Jama: A java matrix package. *URL: http://math. nist. gov/javanumerics/jama*, 2000.

[20] John H Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.

[21] Cay S Horstmann and Gary Cornell. *Core Java 2: Volume I, Fundamentals*. Pearson Education, 2002.

[22] IDEA IntelliJ. the most intelligent java ide. *JetBrains [online].[cit. 2016-02-23]. Dostupné z: https://www. jetbrains. com/idea/# chooseYourEdition*, 2011.

[23] Duan Yuqian He Jiali. Genetic algorithm and its modification [j]. In *PROCEEDINGS OF THE CSU-EPSA*, volume 1, 1998.

[24] Waldemar W Koczkodaj. Statistically accurate evidence of improved error rate by pairwise comparisons. *Perceptual and Motor Skills*, 82(1):43–48, 1996.

[25] Waldemar W Koczkodaj. Testing the accuracy enhancement of pairwise comparisons by a monte carlo experiment. *Journal of Statistical Planning and Inference*, 69(1):21–31, 1998.

[26] Waldemar W Koczkodaj, Marek Kosiek, Jacek Szybowski, and Ding Xu. Fast convergence of distance-based inconsistency in pairwise comparisons. *Fundamenta Informaticae*, 137(3):355–367, 2015.

[27] Waldemar W Koczkodaj and Ryszard Szwarc. On axiomatization of inconsistency indicators for pairwise comparisons. *Fundamenta Informaticae*, 132(4):485–500, 2014.

[28] Waldemar W Koczkodaj and Jacek Szybowski. Pairwise comparisons simplified. *Applied Mathematics and Computation*, 253:387–394, 2015.

[29] Waldemar W Koczkodaj and R Urban. Axiomatization of inconsistency indicators for pairwise comparisons. *International Journal of Approximate Reasoning*, 94:18–29, 2018.

[30] Norman Matloff. *The art of R programming: A tour of statistical software design*. No Starch Press, 2011.

[31] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[32] Melanie Mitchell, Stephanie Forrest, and John H Holland. The royal road for genetic algorithms: Fitness landscapes and ga performance.

In *Proceedings of the first european conference on artificial life*, pages 245–254, 1992.

[33] Katharine M Mullen, David Ardia, David L Gil, Donald Windover, and James Cline. Deoptim: An r package for global optimization by differential evolution. 2009.

[34] Kenneth V Price. Differential evolution: a fast and simple numerical optimizer. In *Proceedings of North American Fuzzy Information Processing*, pages 524–527. IEEE, 1996.

[35] Kenneth V Price. Differential evolution. In *Handbook of Optimization*, pages 187–214. Springer, 2013.

[36] A Kai Qin, Vicky Ling Huang, and Ponnuthurai N Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE transactions on Evolutionary Computation*, 13(2):398–417, 2009.

[37] A Kai Qin and Ponnuthurai N Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *2005 IEEE congress on evolutionary computation*, volume 2, pages 1785–1791. IEEE, 2005.

[38] Jeffrey S Racine. Rstudio: a platform-independent ide for r and sweave. *Journal of Applied Econometrics*, 27(1):167–172, 2012.

[39] Thomas L Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[40] SN Sivanandam and SN Deepa. *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[41] Louis L Thurstone. A law of comparative judgment. *Psychological review*, 34(4):273, 1927.