ADDIS ABABA UNIVERSITY

ADDIS ABABA INSTITUTE OF TECHNOLOGY

SCHOOL OF CIVIL AND ENVIRONMENTAL ENGINEERING

MULTI-OBJECTIVE OPTIMIZATION OF POST-TENSIONED PEDESTRIAN BRIDGE

A Thesis in Structural Engineering

By Metasebia Dabi

September 2021

Addis Ababa

A Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science

The undersigned have examined the thesis entitled "**Multi-objective Optimization of Post-Tensioned Pedestrian Bridge"** presented by **METASEBIA DABI**, a candidate for the degree of **Master of Science,** and hereby certify that it is worthy of acceptance.

Dr. Bedilu Habte		
Advisor	Signature	Date
Dr. Shifferaw Taye		
Internal Examiner	Signature	Date
Dr. Abrham Gebre		
External Examiner	Signature	Date
Dr.-Ing. Mebruk Mohammed		
Chairperson	Signature	Date

UNDERTAKING

I certify that research work titled "**Multi-objective Optimization of Post-Tensioned Pedestrian Bridge**" is my work. The work has not been presented elsewhere for assessment. Where material has been used from other sources, it has been properly acknowledged/referred.

Metasebia Dabi

ABSTRACT

Multi-objective design optimization of post-tensioned box girder pedestrian bridge is presented in this study. The proposed design optimization incorporates some behavior and side constraints specified by the AASHTO Standard specifications. Multi-objective optimization is performed to minimize the cost and maximize the safety factor for the ultimate limit states. Considering stress and deflection limits as design constraints, and geometrical dimensions of the box-girder and shear reinforcements as design variables. Additionally, application of MATLAB routine *gamultiobj* algorithm for the design of post-tensioned pedestrian bridge is presented, and a set of solutions in the pareto front are illustrated in graphical format. Then multi-criteria decision analysis method named TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) is used to determine the single optimal result from the Pareto set solutions by giving weight to the objective functions.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my thesis advisor Dr. Bedilu Habte for being my advisor and for providing support and guidance throughout this thesis. I also want to gratefully acknowledge Dr. Daniel Habtamu for his support and comment throughout the thesis.

I would like to acknowledge Mr. Walter Roberson for his support while working on my design optimization code. I also want to acknowledge all MATLAB central community members for their comments.

Finally, I am grateful to my parents for their help, support, and encouragement in accomplishing the thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1 INTRODUCTION

1.1 Background

A bridge is a type of structure that generally provides passage over an obstacle without closing the way beneath. The main concept of the prestressed concrete bridge was introduced by Freyssinet in the year 1904; he attempted to introduce permanently acting force in concrete to resist the elastic forces developed under loads and this idea was later developed under the name of prestressing. Prestressing refers to the permanent internal stress in a structure to improve performance by reducing the effect of external forces. The compression performance of concrete is strong but its tension performance is weak. The main idea of prestressing concrete is to counteract the tension stresses that are induced by external forces, (Krishna, 2010).

A box girder bridge is a bridge in which the main beams comprise girders in the shape of a hollow box. It is formed when two web plates are joined by a common flange at both the top and the bottom. Box girders are more suitable for larger spans and wider decks. They are elegant and slender. Economy and aesthetics further lead to the evolution of cantilevers in top flanges and inclined webs in the external cell of the box girder, and the dimensions can be controlled by pre-stressing, (Rama et al., 2010).

Optimum design is an alternative to the conventional design method. It normally implies the most economic structure without impairing the functional purposes of the structure. An optimization technique transforms the conventional design process of trial and error into a formal and systematic procedure that yields a design that is best in terms of the design's merits. It is a completely automated process that allows lesser skilled and experienced engineers to create an optimum design. It forces the designer to identify the design variables that describe the design of the structure and also an objective function that measures the relative merit of feasible designs and constraints on performance that the design must satisfy, (Hassanain and Loov.,2003)

Optimization is a procedure of finding and comparing feasible solutions until no better solution can be found. The solution is termed good or bad in terms of an objective, which is often the cost of fabrication, amount of harmful gases, the efficiency of a process, product reliability, or other factors. A significant portion of research and application in the field of optimization considers a single objective, although most real-world problems

involve more than one objective. The presence of multiple conflicting objectives (such as simultaneously minimizing the cost of fabrication and maximizing product reliability) is natural in many problems and makes the optimization problem interesting to solve since no one solution can be termed as an optimum solution to multiple conflicting objectives, the resulting multi-objective optimization problem resorts to several trade-off optimal solutions, (Deb, 2001).

Cost optimization for post-tensioned I girder bridge was performed, considering the number of strands, web width, the thickness of the slab, concrete strength as a design variable. Cost including material, installation, and fabrication cost is considered as a design objective. Concluded that optimum girder spacing is found for smaller span and that the concrete strand is much higher in high strength than low concrete strength, (Shovel R.,2010).

Optimization of prestressed concrete precast pedestrian bridges made of a precast concrete beam that integrates an upper reinforced concrete slab for pedestrian traffic using threshold accepting (TA) algorithms. The algorithms are applied to a typical pedestrian Bridge of 40 m of span length and 6.00m of width. Finally, the solution and run times indicate that a heuristic optimization is a forthcoming option for the design of real prestressed structures, (Marti et al., 2009).

The genetic algorithm method is used for the optimization of the I-Girder Bridge and design is carried out by the limit state method. A computer program is coded using MATLAB CSI-API software for optimization and analysis results are retrieved from CSI Bridge V20 software to carry out the optimization process (Bhinge and Fernandes., 2018). Finally, they conclude that the pre-stressing force is reduced for the optimized section compared to the initial section and the genetic algorithm is adequate for the optimization of the I-girder bridge, (Bhinge and Fernandes., 2018).

Great major's research in structural design optimization uses the two general approaches of multi-objective optimization. One is to combine the individual objective function into a single composite function or move all but one objective to the constraint set. In the former case, determination of a single objective is possible with methods such as utility theory, weighted sum method, etc. but the problem lies in the proper selection of the weights or utility function to characterize the decision maker’s preferences. It is difficult to precisely

and accurately select weights, even for someone familiar with the problem domain. Corresponding to this drawback is that scaling among objectives is needed and small perturbation in the weights can sometimes lead to a quite different solution. (Konak et al., 2006).

The constraint approach method is used to transform the multi-objective optimization into a single-objective optimization problem, Lounis and Cohn, (Lounis and Cohn.,1993), studied the design of a post-tensioned floor slab and a pre-tensioned highway bridge system for two conflicting objectives: minimum cost and minimum initial camber were conducted and the advantage of the e-constraint approach over other approaches for transforming the multi-objective optimization problem into a single objective problem was studied. Finally, the Pareto optimum achieves a compromise between the two conflicting objectives and represents more rational solutions than those obtained by independently optimizing each objective function (Lounis and Cohn. ,1993).

The second general approach is to determine an entire Pareto optimal solution set or a representative subset. A Pareto optimal set is a set of solutions that are non-dominated concerning each other. while moving from one Pareto solution to another. There is always a certain amount of sacrifice in one objective to achieve a certain amount of gain in the other. A reasonable solution to a multi-objective problem is to investigate a set of solutions, each of which satisfies the objectives at an acceptable level without being dominated by any other solution, (Konak et al., 2006).

In real-world applications, most optimization problems involve more than one objective to be optimized. the objectives in most of the engineering problems are often conflicting, i.e., maximize performance, minimize cost, maximize reliability, etc. In this case, one extreme solution would not satisfy both objective functions and the optimal solution of one objective will not necessarily be the best solution for the other objectives. Therefore, different solutions will produce trade-offs between different objectives, and a set of solutions is required to represent the optimal solution of all objectives, (Lamont et al., 2004).

1.2 Significance of the Study

Furthermore, in structural optimization, one of the major difficulties inherent in solving realistic engineering problems is the selection of meaningful objective function that includes all relevant criteria with adequately chosen weighting factors for a given structural design problem. This problem can be overcome by considering the multi-objective optimization strategy. Multi-objective optimizations (MOO) of structural systems considering several objectives simultaneously are practically feasible and more informative and beneficial than single-objective optimizations. In MOO problems the designer tries to find value for the design variables which optimize the objective functions simultaneously.

Therefore, this research investigates the application of MATLAB routine *gamultiobj* algorithm for the design of post-tensioned box girder pedestrian bridge, taking into account two design objectives which are minimizing cost and maximizing safety factor for the ultimate limit state. The design variables are width, thickness, depth of the girder and shear reinforcement while the design constraints are normal stress in concrete, the stress in steel tendon, the shear stress in concrete and deflection at the mid-span. Finally, a design optimization is performed and results are expressed in graphical format. Then from results obtained by using TOPSIS single optimal result is taken as final result.

1.3 The Objective of the Study

1.3.1 General Objective

The main objective of the study is to investigates the applicability of MATLAB routine *gamultiobj* algorithm, taking into account two design objectives which are minimizing cost and maximizing safety factor for the ultimate limit state. Satisfying all the design requirements of shear, flexure, and deflection for the post-tensioned box-girder pedestrian bridge.

1.3.2 Specific- Objectives

The specific objectives of this study are the following:

- Developing multi-objective optimization outline for post tensioned box girder bridge.

- To numerically investigate the applicability of MATLAB routine *gamultiobj* algorithm for designing a post-tensioned box girder pedestrian bridge.
- To minimize the cost and maximize the safety factor for ultimate limit state of a simply-supported post-tensioned box girder pedestrian bridge.
- To demonstrate the applicability of TOPSIS in selecting results from the pareto set solutions.

1.4 Scope of the Study

The present study develops a multi-objective optimization design optimization code for the post-tensioned pedestrian bridge using *gamultiobj* algorithm which is one of the best performers among the multi-objective optimization MATLAB toolbox, taking into account two design objectives which are minimizing the material cost of the structure and maximizing the safety factor for the ultimate limit state of the structure. The design variables are the thickness of the top and bottom flange, the thickness of the web, overhang width, depth of the girder, width of the bottom flange, and area of transversal reinforcement while the design constraints are normal stress in concrete, the stress in steel tendon, the ultimate flexural strength for flexure, the shear stress in concrete and deflection at the mid-span. Finally, a computer program will be developed and the mathematical description of the multi-objective optimization will be expressed using a Pareto chart.

1.5 Methodology

To meet the objective of this study literature survey about a prestressed concrete bridge, structural optimization, and MATLAB toolbox algorithms have been conducted.

MATLAB code is developed for designing a prestressed concrete bridge and the design code is verified by performing designs examples with simplified hand calculation, after validating the design, optimization code for simple cantilever welded beam using gamultiobj is prepared to check the algorithm with simple structure then verify code developed using the results obtained from previous works. then the design code for the prestressed box girder pedestrian bridge considering the two objectives, many variables, linear and nonlinear constraints is developed. Finally, optimum results are displayed in graphical format, and from this Pareto set solution in the graph, the study shows the method for selection of a single optimal solution by giving weight to the objective functions using a method named TOPSIS.

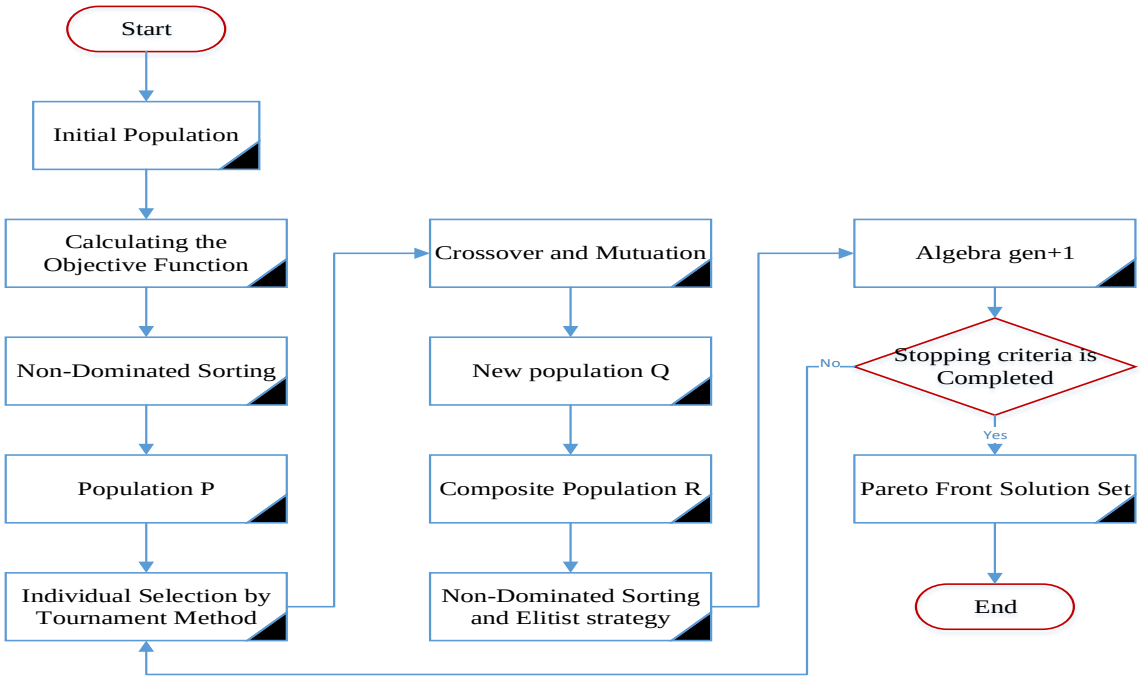


Figure 1-1 Flowchart for Multi-Objective Optimization using Genetic Algorithm (MathWork, 2018)

1.6 Organization of the Thesis

The thesis has been divided into six chapters

Chapter 1: describes the introductory part that includes the background, significance, objective, scope, and methodology of the thesis.

Chapter 2: describes a literature review on the field of Prestressed Bridge structures, structural optimization, single and multi-objective optimization.

Chapter 3: describes design considerations for the box girder bridge, gamultiobj Algorithm, and verification of the developed MATLAB code using simply supported cantilever reinforced concrete beam is performed.

Chapter 4: describe the detail of the objectives, design constraints, and design variables that are used in the study to develop the general design framework of the post-tensioned box girder bridge. Optimization results from MATLAB 2018a are presented in graphical format and the results are discussed in detail.

Chapter 5: presents the conclusion of the study and also provides recommendations for future study.

CHAPTER 2 LITERATURE REVIEW

2.1 Box Girder Bridge Structures

Box girders are used in buildings and bridge, most of the time they are used in bridge structures. They are universally applied from the point of view of load carrying, to their indifference as to whether the bending moment is positive or negative, and to their torsional stiffness; from the point of view of the economy (Rama et al., 2010).

Advantage of Box Girders Bridge

Box girder bridge has many importance, some are discussed below: -

- Single or multicell reinforced concrete box girder bridges have been widely used as an economic and aesthetic solution for bridge structures.
- Box girder bridges have large torsional rigidity, and provide more aesthetically pleasing.
- The interior of the box girder bridge can be used to accommodate services such as gas pipes, water mains, etc.
- For a longer span, the bottom flange could be used as another deck accommodates traffic.
- It has high structural efficiency which minimizes the pre-stressing force required to resist a given bending moment, and its great torsional strength with the capacity this gives to re-center eccentric live loads, minimizing the prestress required to carry them.

2.2 Prestressed Structures

Prestressed concrete is a particular form of reinforced concrete, prestressing involves the application of an initial compressive load to the structure to reduce or eliminate the internal tensile forces and thereby control or eliminate cracking. The initial compressive load is imposed and sustained by highly tensioned steel reinforcement (tendons) reacting on the concrete, with cracking reduced or eliminated, a prestressed concrete section is considerably stiffer than the equivalent reinforced concrete section. Prestressing may also impose internal forces that are opposite signs to the external loads and may therefore significantly reduce or even eliminate deflection.

Prestressing is a transformation of concrete into an elastic material this concept treats concrete as an elastic material and is probably still the most common viewpoint among engineers. It is credited to engineer Freyssinet who visualized prestressed concrete as essentially concrete which is transformed from a brittle material into an elastic one by the pre-compression given to it. Concrete which is weak in tension and strong in compression is compressed (generally by steel under high tension) so that the brittle concrete would be able to withstand tensile stresses. Therefore, no tensile stress in the concrete, there can be no cracks and the concrete is no longer a brittle material but it became an elastic material (Line and Ned, 1981).

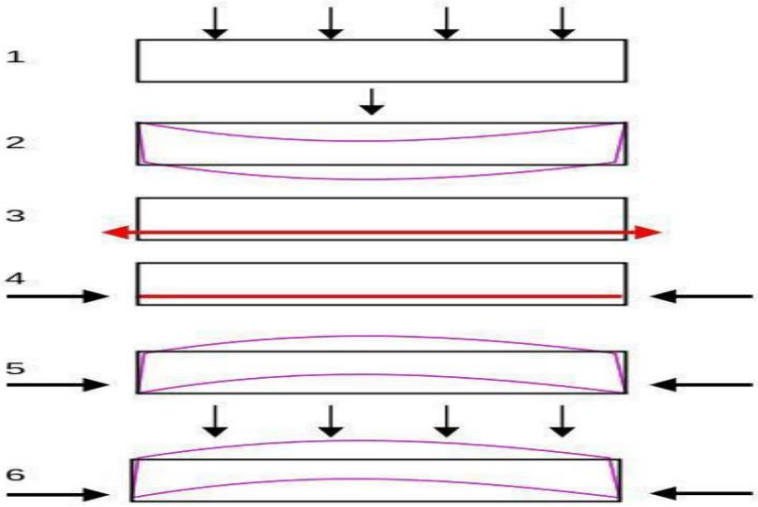


Figure 2-1 Behavior of RC Member with and without Prestressing (Hurst, 1998)

2.2.1 Prestressing System

There are two types of prestressing systems: pre-tensioning and post-tensioning systems. Pre-tensioning systems are methods in which the strands are tensioned before the concrete is placed. This method is generally used for the mass production of linear concrete members. Pre-tensioning can neither be used to connect two precast concrete components nor connect precast components to cast in place concrete members (Hurst, 1998).

Post-tensioning systems are methods in which the tendons are tensioned after the concrete has reached a specified` strength. This technique is often used in projects with a very large cast in place elements on falsework. The main advantage of post-tensioning is its ability to post-tension both precast and cast-in-place members. In post-tensioning, the tendons are

stressed after the concrete is cased and hardened to certain strength to withstand the prestressing force. The tendons are either coated with grease or encased with flexible metal before placing in forms to prevent the tendons from bonding to the concrete to concrete during placing and curing of concrete. In the first case, the tendons are bounded with the concrete, and corrosion is prevented, so is called bonded system whereas in the latter case and the tendons are not bounded with concrete this system is called an unbonded system (Hurst, 1998).

2.2.2 Prestress Loss

The initial prestress in the concrete undergoes a gradual reduction with time from the stage of the transfer due to various causes. It is generally referred to as the reduced tensile stress in the tendons after stressing. Prestress losses are divided into two categories which are instantaneous losses (losses due to anchorage set, friction between tendons and surrounding concrete, and elastic shortening of concrete during construction stage) and Time-dependent losses (including losses due to shrinkage, creep, and relaxation of steel during the service life (Krishenan, 2010).

- Anchorage set Loss: - is an effect of anchorage set on the cable stress. this loss occurs in the vicinity of the jacking end of post-tensioned members as the post-tensioning force is transferred from the jack to the anchorage block.

- Friction Loss: - are caused by the tendon profile curvature effect and local deviation in tendons profile wobble effects. When a tendon is jacked from one or both ends the stress along the tendon decreases away from the jack due to the effects of friction. Friction loss occurs only in post-tensioned members.

- Elastic Shortening Loss: - loss occurs when a prestress is transferred to the concrete the member itself shortens and the prestressing steel shorten with it. Due to this, there is a loss of prestress in the steel.

- Shrinkage Loss: - is a loss due to shrinkage of concrete in which the concrete contract. It is dependent on time and moisture conditions, and the amount of shrinkage varies widely.

- Creep Loss: - creep is the property of concrete by which it continues to deform with time under sustained load at unit stresses within the accepted elastic range. It is a time-dependent phenomenon. Creep of concrete results in a loss in steel stress.

- Relaxation of Steel Loss: - is a loss of steel under nearly constant strain at a constant temperature.

2.3 Optimization Method

Traditionally, technical guides and recommendations have on condition that is a starting point for the design process. There are strict codes to assure an appropriate reliability and safety level. Concrete bridge design codes have been developed by the European Committee for standardization (European Standard, 1992) , and American Association of state highway and transport officials (AASHTO, 2012), the emergency of optimization methods to minimize the structural weight and economic cost has enabled designers to explore new design forms.

Optimization refers to finding one or more feasible solutions which correspond to extreme values of one or more objectives, the need for finding such optimal solutions in a problem comes mostly from the extreme purpose of either designing a solution from the minimum possible cost of fabrication, or for maximum possible reliability, or others. Because of such extreme properties of optimal solutions, optimization methods are of great importance in practice, particularly in engineering design, scientific experiments, and business decision-making (Deb, 2001).

There are two kinds of optimization depending on the number of objectives involved in the problem which are single-objective and multi-objective optimization. When an optimization problem involves only one objective function, the task of finding the optimal solution is called single-objective optimization, whereas if an optimization problem involves more than one objective function it is called multi-objective optimization. Most real-world search and optimization problems naturally involve multiple objectives. In multi-objective optimization, different solutions may produce trade-offs (conflicting scenarios) among different objectives. A solution that is extremely concerning one objective requires a compromise in other objectives (Deb, 2001).

2.3.1 Multi-Objective Optimization

The multi-objective optimization problem (MOOP) deals with more than one objective function. In most practical decision-making problems. Multiple objectives or multiple criteria are evident. Because of a lack of suitable solution methodologies, a MOOP has been mostly cast and solved as a single-objective optimization problem in the past. However, there exist several fundamental differences between the working principles of the single and multi-objective optimization algorithm. In a single–objective optimization

problem, the task is to find one solution which optimizes the given objective function. Extending the idea to multi-objective optimization, it may be wrongly assumed that the task in a multi-objective optimization is to find an optimal solution corresponding to each objective function.

Multi-Objective Optimization Problem

A multi-objective optimization problem has a number of objective functions which are to be minimized or maximized. As in the single-objective optimization problem ,here too the problem usually has a number of constraints which any feasible solution (including the optimal solution) must satisfy. Multi-objective optimization has the following general form: Minimize/Maximize $fm(x)$, $m = 1,2, \dots \dots, M;$

$$gj(x) \geq 0 \qquad j = 1,2, \dots \dots \dots J;$$

$$hk(x) = 0 \qquad k = 1,2, \dots \dots, k$$

$$X_i^{(L)} \leq xi \leq X_i^{(L}, \qquad i = 1,2, \dots \dots, n$$

Where a solution x is a vector of n decision variables: $x= x_1,x_2,x_3,\dots.x_n$. The last set of constraints are called variable bounds, restricting each variable x with in the upper and lower bound. In multi-objective optimization there are two bounds constitute a decision variable space and an Objective space. Bounds of the design variables constitute decision variable space. The problem has J in equality and K equality constraints. The terms $gj(x)$ and $hk(x)$ are called constraint functions. The constraints must convert in to greater -than-equal-to type constraint by multiplying the constraints function by -1.

In multi-objective optimization the objective function constitute multi-dimentional space,in addition to the usual decision variable space .this additional space is called the objective space, Z, for each solution x in the decision variable space, there exist a point in the objective space .there are M objective functions $f(x) =\{ f_1(x) , f_2(x)\dots,f_M(x)\}^T$,each objective functions can be either minimized or maximized. In optimization suggests that to convert a maximization problem into minimization should multiply the objective function by -1.

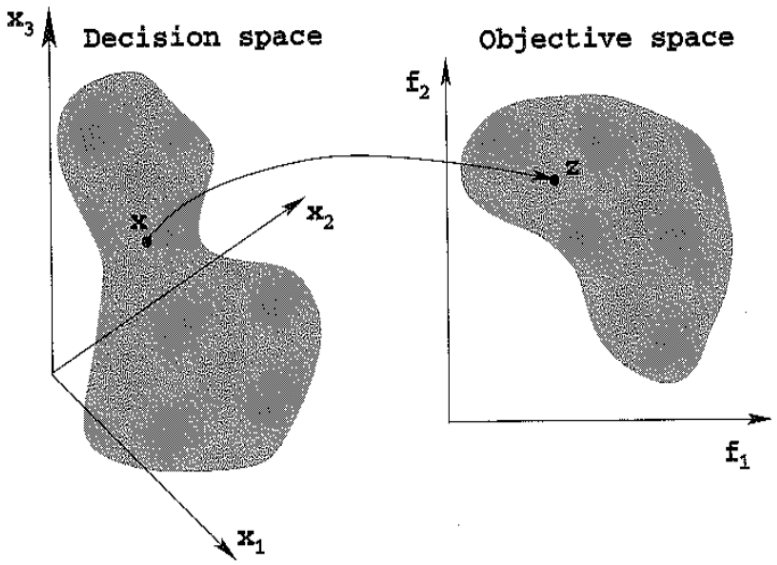


Figure 2-2 Representation of the decision variable space and the corresponding objective (Deb, 2001)

2.3.2 Classification of Multi-Objective Optimization Methods

According to Deb (Deb, 2001), there are two methods of multi-objective optimization which are discussed below:

2.3.2.1 Classical multi-objective optimization methods

Classical multi-objective optimization methods have also been around for at least the past four decades. During this period, there were many algorithms according to various considerations. It is classified into the following two types which are generating and preference-based methods. The classical multi-objective optimization based on the preference base is discussed below.

In classical multi-objective optimization algorithms, most algorithms convert MOOP problems into single-objective optimization by using some user-defined procedures. Of these, the weighted approach converts multiple objectives into a single objective by using the weighting sum of objectives. The weight vector is user-defined since this method is incapable of finding a trade-off optimal solution in a problem with a non-convex Pareto-optimal region, the ε-Constraint approach converts all by one of the objective functions into constraints. This method also requires a weight vector for emphasizing the objective differently. For some of these algorithms, there exist theorems proving that the optimal solution of the converted single-objective optimization problem is one of the Pareto-optimal solutions. Moreover, each of the above classical methods involves several user-

defined parameters, which are difficult to set in an arbitrary problem. There are several classical methods in order of increasing use of preference information such as Weighted Sum Method, ε-Constraint Method, Weighted Metric Method, Bonson's Method, Value Function Method…… etc.

2.3.2.2 Evolutionary Multi-objective Optimization methods

The main motivation for using evolutionary algorithm is to solve multi-objective optimization problems is because evolutionary algorithm deal simultaneously with a set of possible solutions (the so-called population) which allows us to find several members of the Pareto optimal set in a single run of the algorithm, instead of having to perform a series of separate runs as in the case of the traditional mathematical programming techniques. Additionally, EAs are less susceptible to the shape or continuity of the Pareto front (Coelio and Lamont, 2004).

Hybrid harmony search for sustainable design of post-tensioned concrete box-girder pedestrian bridges, using harmony search algorithm combining threshold optimization is used to find the geometry and the materials for which the sum of the costs of the emission is the lowest, yet satisfying the requirements for structural safety and durability. The finding indicates that both objectives lead to similar cost results (Segura et al., 2015).

2.4 Multi-objective optimization using metaheuristics

Optimization Problems with continuous variables gave infinite possible solutions but only one global optimum. The process of finding a solution for such a problem can be seen as a problem of search. In search-based methods, one randomly searches different regions of the search space while being guided towards the optimum point by some mechanism. This allows for the search method not to spend too much time searching for solutions in regions where there is no potential global optimum. Meta-heuristic methods can also be applied to problems that have a discrete search space. Some algorithms that fall into this category are simulated annealing (SA), Swarm algorithms, artificial neural networks (ANNs), and genetic algorithms(GAs). All the above algorithms have been successfully applied in structural design and optimization problems, but in this study, only genetic algorithm is discussed in detail (Camp et al., 1998).

All meta-heuristic methods have the following properties in common:

- All start at a random point (or several random points) in the search space
- All have probabilistic convergence behavior
- All have mechanisms that allow them to propagate through large separate quadrants of the search space in parallel.

2.5 Genetic Algorithms

The concept of the genetic algorithm was developed by Holland and his colleagues in the 1960s and 1970s. Genetic algorithms are inspired by the evolutionist theory explaining the origin of species. this algorithm works with the low survival of the fittest. the one which is strongest than the others has a great probability to pass its genes to the future by reproduction. The species carrying the correct combination in the genes became dominant in the population. During the process of evolution, changes are expected to happen in the genes. So, a new species evolves from the old ones, and unsuccessful changes are eliminated by natural selection (Konak. et al., 2006).

In genetic algorithm terminology, a solution vector $x \in X$is called an individual. Individuals are made of discrete units called genes. A collection of these individuals is called population. The population is normally randomly initialized. As the search evolves, the population includes fitter and fitter solutions.

Genetic algorithms use two operators to generate new solutions from existing ones: crossover and mutation. The crossover operator is the most important operator of the genetic algorithm. In a crossover, generally, two chromosomes, called parents, combined to form new chromosomes called offspring. by iteratively applying the crossover operator, genes of good chromosomes are expected to appear more frequently in the population, which leads to convergence of good solutions. The mutation operator introduces random change into characteristics of chromosomes. In typical genetic algorithm implementations, the mutation rate is very small and depends on the length of the chromosome. Mutation plays a critical role in the genetic algorithm. As discussed earlier, crossover leads the population to converge by making the chromosomes in the population alike. Whereas mutation reintroduces genetic diversity back into the population and assists the search escape from local optima (Konak et al., 2006).

Design components of multi-objective GA

According to (Konak et al., 2006), the basic design components of the multi-Objective genetic algorithm are discussed below

- **Fitness function**

Weighted sum approach: - an approach to solve multi-objective optimization by assigning a weight to each objective function so that the problem will be converted into a single objective problem.

Altering objective functions: - populations are randomly divided into K equal-sized sub-populations and each solution in the sub-population is assigned a fitness value based on the objective function. This approach is a straightforward extension of a single objective GA to solve multiple problems.

Pareto-ranking approaches: - the population is ranked according to a dominance rule, and then each solution is assigned a fitness value based on its rank in the population. Objectives are assumed to be minimized. Therefore, a lower rank corresponds to a better solution.

- **Diversity**

To obtain solutions over the Pareto front it is important to maintain a diverse population in multi-objective genetic algorithms. Several approaches for diversifying a population are discussed below: -

Fitness sharing:- it artificially reduces the fitness of solutions in densely populated areas and encourages the search in unexplored sections. Densely populated areas will be identified and a penalty method is used to penalize the solution located in such areas.

Crowding distance:- a measure of solution around a population is computed without requiring a user-defined parameter .and when two solutions are in the same non-dominated front, a solution with a higher crowding distance is the winner.

Crowding distance:- a measure of solution around a population is computed without requiring a user-defined parameter .and when two solutions are in the same non-dominated front, a solution with a higher crowding distance is the winner.

- **Elitism**

It means the best solution survives to the next generation. All non-dominated solutions obtained from multi-objective optimization are elite solutions. Multi-objective optimization uses two strategies to implement elitism.

Maintain elitist solution in population: - it is a straightforward implementation of elitism in a multi-objective which is a copy of all non-dominated solutions in solution to the next generation and fill the rest by dominated solutions.

Elitism with external population: - storing non-dominated solutions identified in the search process. then selection of elitist solutions from a store and reintroduced to the population is performed.

- **Constraint handling**

In real-world optimization design problems, many constraints must be satisfied. In single-objective optimization, four different constraint handling strategies are used which are the death penalty, reducing infeasible solution fitness value by penalty function, crafting genetic operator, and transferring infeasible solutions to feasible ones. In multi-objective optimization, it is not a straightforward method like single objective optimization because fitness assignment is based on the non-dominance rank of the solution.

Niched selection strategy: - for solutions which are both feasible or infeasible by taking random reference at x among the feasible or infeasible solutions, comparing the solutions or results in set x. if one of them is dominated by at least one solution, a winner is formed otherwise the minimum infeasible or feasible one will be selected by the tournament selection by counting the smaller nich count.

Constraint tournament method: - two solutions are randomly chosen from the population. The winner is the one in a more preferred non-constrain dominance front. This method is advantageous because it can easily be integrated into a multi-objective.

CHAPTER 3 METHODOLOGY

3.1 Prestressed Box Girder Design Considerations

In this study, the design for the prestressed bridge is performed according to the design code and the following design criteria are considered to perform the design of the bridge structure.

3.1.1 Basic Theory

Prestressed concrete has the following distinguishing characteristics

- The stress for concrete and prestressing steel and deformation of the structure at each stage, are investigated based on the elastic theory
- The prestressing force is determined by concrete stress limits under service load
- For prestressed concrete flexural members, the stress at various service load stages can be expressed by the following formula:
- $f = \frac{P_j}{A} \pm \frac{P_j e y}{I} \pm \frac{M y}{I}$ 3-1

3.1.2 Stress Limits

The stress limits are the basic requirements for designing a prestressed concrete member. The purpose of the stress limit on the prestressing tendon is to mitigate tendon fracture, avoid inelastic tendon deformation, and allow for pre-stress losses.

3.1.3 Flexural Resistance

Flexure resistance in the strength limit state is based on the following assumptions (AASHTO, 2012):

For members with bounded tendons, strain is linearly distributed across a section. For a member with unbounded tendons, the total change in tendon length is equal to the total change in member length over the distance between two anchorage points. The tensile strength of concrete is neglected.

Non-prestressed reinforcement reaches the yield strength and the corresponding stresses in the prestressing tendons are compatible with plane section assumptions.

For a member with a flange section subjected to uniaxial bending, the equation of equilibrium are used to obtain a nominal moment resistance of:-

$$M_n = A_{ps}f_{ps}\left(d_p - \frac{a}{2}\right) + A_s f_s\left(d_s - \frac{a}{2}\right) - A_s^{'}f_s^{'}\left(d' - \frac{a}{2}\right) + 0.85f_c^{'}(b - b_w)h_f\left(\frac{a}{2} - \frac{h_f}{2}\right) \quad 3\text{-}2$$

$$a = B_1 c \quad 3\text{-}3$$

For bonded tendons:

$$c = \frac{A_{ps}f_{ps} + A_s f_s - A_s^{'}f_s^{'} - 0.85f_c^{'}(b - b_w)h_f}{0.85B_1 f_c^{'}b_w + kA_{ps}\frac{f_{pu}}{d_p}} \geq h_f \quad 3\text{-}4$$

$$f_{ps} = f_{pu}\left(1 - k\frac{c}{d_p}\right) \quad 3\text{-}5$$

$$k = 2\left(1.04 - \frac{f_{py}}{f_{pu}}\right) \quad 3\text{-}6$$

$$0.85 \geq B_1 = 0.85 - (f_c^{'} - 4)0.05 \geq 0.65 \quad 3\text{-}7$$

Where: A represent area; f is the stress; b is the width of the compression face of the member; $b_w$is the web width of a section; h_f is the compression flange depth of the cross-section; $d_p$and $d_s$are the distance from the extreme compression fiber to the centroid of prestressing tendons and the centroid of tension reinforcement, respectively; subscript c indicates the specified strength for concrete, and p and s mean prestressing steel and reinforcement steel respectively.

For unbonded tendons

$$c = \frac{A_{ps}f_{ps} + A_s f_s - A_s^{'}f_s^{'} - 0.85f_c^{'}(b - b_w)h_f}{0.85B_1 f_c^{'}b_w} \geq h_f \quad 3\text{-}8$$

Minimum reinforcement limit:

$$\phi M_n \geq 1.2M_{cr} \quad 3\text{-}9$$

In which ϕ is the flexural resistance factor 1 for prestressed concrete and $M_{cr}$is the cracking moment strength given by the elastic stress distribution and the modulus of rupture of concrete.

$$M_{cr} = \frac{I}{y_t}(f_r + f_{pe} - f_d) \quad 3\text{-}10$$

Where f_{pe} is the compressive stress in concrete due to effective prestresses and f_d is the stress due to the factored selfweight.

3.1.4 Shear resistance

The nominal shear resistance at the strength limit state is contributed by tensile stress in the concrete and tensile stresses in the transverse reinforcement and vertical component of prestressing force. It shall be determined by the formula (AASHTO 2012):

$$\text{Vn= the lesser of } \begin{cases} V_c + V_s + V_p \\ 0.25 f_c^{'} b_v d_v + V_p \end{cases} \quad 3\text{-}11$$

Where

$$V_c = 0.0316 B \sqrt{f_c^{'}} b_v d_v \quad 3\text{-}12$$

$$V_s = \frac{A_v f_y d_v (cos\,\theta + cot\,\alpha)\, sin\,\alpha}{s} \quad 3\text{-}13$$

Where: b_v is effective web width determined by subtracting the diameters of un-grouted ducts or one-half the diameters of grouted ducts; d_v is the effective depth between the resultants of the tensile and compressive forces due to flexure, $A_v$is the shear reinforcement within distance s;$s$is the spacing of the stirrups; $\alpha$is the angle of inclination of shear reinforcement to the longitudinal axis; $B$is a factor indicating the ability of diagonally cracked concrete to transmit tension; θ is the angle of inclination of diagonal compressive stresses. The values of B and θ is determined by:

$$\beta = \frac{4.8}{1 + 750\varepsilon_s} \quad 3\text{-}14$$

$$\theta = 29 + 3500\varepsilon_s \quad 3\text{-}15$$

$$\varepsilon_s = \frac{\frac{|M_u|}{d_v} + 0.5N_u + |V_u - V_p| - A_{ps} f_{po}}{E_s A_s + E_p A_{ps}} \quad 3\text{-}16$$

Where: $|M_u|$is the absolute value of the factored moment, not to be taken into account if it is less than where $|V_u - V_p| d_v$;$N_u$is the factored axial force associated with the factored shear force V_u; $f_{po}$is the stress in prestressing steel when the stress in the surrounding concrete is zero and can be taken as $0.7 f_{pu}$; $V_p$is the component in the direction of the applied shear of the effective prestressing force.

3.1.5 Camber and Deflections

Camber has reversed deflection and is caused by prestressing. A careful evaluation of camber and deflection for a prestressed concrete member is necessary to meet serviceability requirements. The following formulas are used to determine the midspan camber of simply supported members (AASHTO, 2012).

For straight tendon

$$\Delta = \frac{L^2}{8E_cI}M_e \qquad 3\text{-}17$$

For a one-point harping tendon

$$\Delta = \frac{L^2}{8E_cI}\left(M_c + \frac{2}{3}M_e\right) \qquad 3\text{-}18$$

For a two-point harping tendon

$$\Delta = \frac{L^2}{8E_cI}\left(M_c + M_e - \frac{M_e}{3}\left(\frac{2a}{L}\right)^2\right) \qquad 3\text{-}19$$

For parabolic tendon

$$\Delta = \frac{L^2}{8E_cI}\left(M_e + \frac{5}{6}M_c\right) \qquad 3\text{-}20$$

Where: $M_e$is the primary moment in the end, and $M_c$is the primary moment at mid-span. uncracked gross sectional properties are often used in calculating camber.

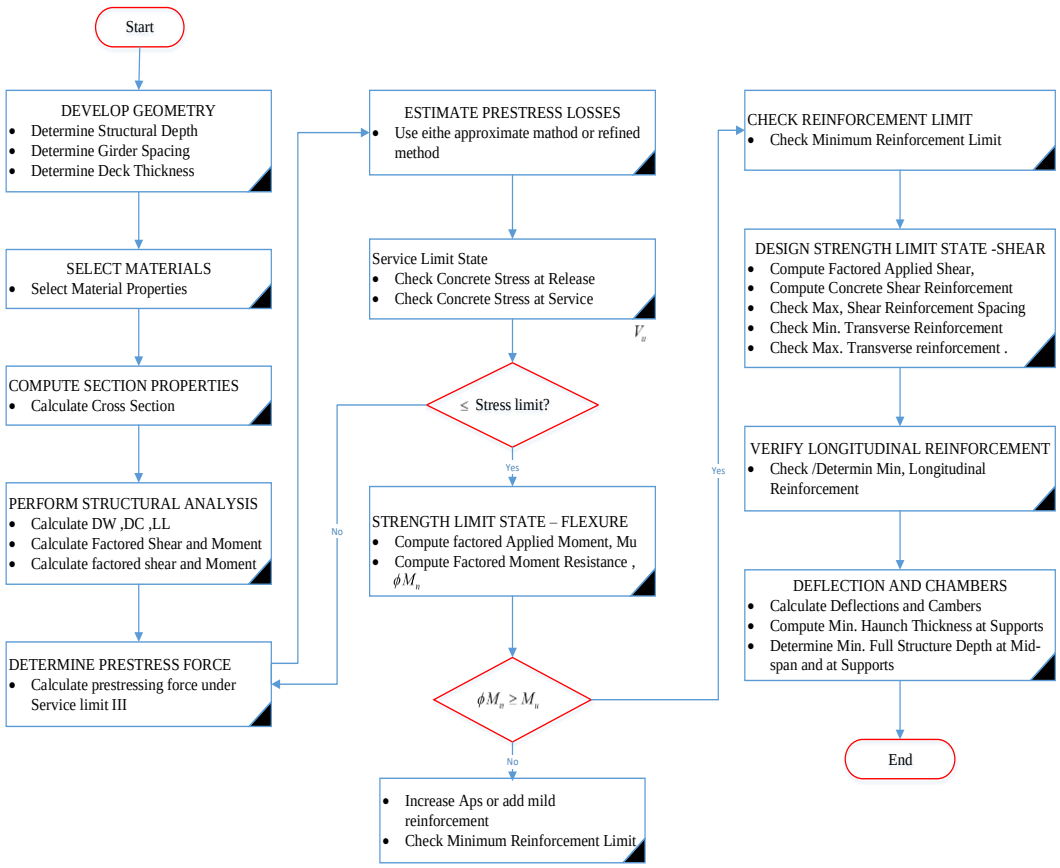


Figure 3-1 Flow Chart for Prestressed Bridge Design (Wai-Fah and Lian , 2014)

3.2 Genetic Algorithm

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that are based on natural selection, the process that drives biological evolution. The genetic algorithm repeatedly modifies a population of individual solutions. At each step, the genetic algorithm selects individuals at random from the current population to be parents and uses them to produce the children for the next generation. Over successive generations, the population "evolves" toward an optimal solution. You can apply the genetic algorithm to solve a variety of optimization problems that are not well suited for standard optimization algorithms, including problems in which the objective function is discontinuous, non-differentiable, stochastic, or highly nonlinear (MathWork, 2018).

The genetic algorithm uses three main types of rules at each step to create the next generation from the current population: (MathWork, 2018).

- Selection rules select the individuals, called parents, that contribute to the population at the next generation.
- Crossover rules combine two parents to form children for the next generation.
- Mutation rules apply random changes to individual parents to form children.

3.3 Gamultiobj Algorithm

Gamultiobj is an algorithm used in MATLAB toolbox for multi-objective optimization using a controlled elitist genetic algorithm, an elitist GA always favors individuals with better fitness value (rank). A controlled elitist GA also favors individuals that can help increase the diversity of the population even if they have a lower fitness value.

Multi-objective Terminology

Most of the terminology for this algorithm is the same as the "Genetic Algorithm" terminology. For more details about the terminology and the algorithm, (Deb, 2001).

- Dominance: - the term dominance is equivalent to the term inferior x dominates y exactly when y is inferior to x

- Rank: - for a feasible individual, there is an iterative definition for the rank of an individual. Rank 1 individuals are not dominated by any other individual. Gamultiobj uses rank to select parents.

- Crowding Distance: - the crowding distance is a measure of the closeness of an individual to its nearest neighbors. The algorithm measures distance among individuals of the same rank. individuals of the same rank with a higher distance have a higher chance of selection.

- Spread: - It is used in a stopping condition. iteration halt when the spread does not change much, and the final spread is less than an average of recent spreads.

Steps in gamultiobj algorithm

Gamultiobj algorithm follows the following steps to perform.

- Initialization: - the population can give a random initial population by using the initial population matrix. The number of individuals in the population is set to the value of the population size option. By default, the algorithm creates a population that is feasible for bounds and constraints. It evaluates the objective function and constraints for the population and uses those values to create scores for the population.

- Iterations: - the main iteration of the algorithm proceeds by selecting parents for the next generation using the selection function on the current population. The function available for the algorithms could be custom selection function or binary tournament. Then new population is created by using crossover and mutation , score is given to the children , and rank is given to the extended population based on the score and the crowding distance for all individuals.

- Stopping Conditions: - the algorithm terminates when a geometric average of the relative change in the value of the spread over options, the maximum number of generations exceeded, optimization terminated by an output function, and when there is no feasible point found.

3.4 Technique for Order Preference by Similarity to Ideal Solution (TOPSIS)

TOPSIS method is a multi-criteria decision analysis method, which was developed by Chick-Lai, Yoon, and Hwang in 1981 with further developments by Yoon in 1987, and Hwang, Lai, and Liu in 1993, (Hwang, et al.,1994). Establish that the chosen solution should have the shortest distance to the ideal solution I^+ and the longest distance from the nadir solution.

The following procedures are used to use this method:

Step 1: Calculate the Normalization Matrix of the Objectives

$$\bar{X}_{ij} = \frac{X_i}{\sqrt{\sum_i^m X_i^2}} \qquad \textbf{3-21}$$

Step 2: Calculate Weighted Normalized Matrix

$$V_{ij} = \bar{X}_{ij} * W_j \qquad \textbf{3-22}$$

Step 3: Calculate the Ideal Best Value and Worst Value

- It is the minimum or the maximum value of the objective function expressed as V_{ij}^{+} and V_{ij}^{-} means ideal best and worst value respectively.

Step 4: Calculate the Euclidian Distance from the Ideal Best

$$S_i^{+} = \left[\sum_j^m \left(V_{ij} - V_{ij}^{+}\right)^2\right]^{0.5} \qquad \textbf{3-23}$$

Step 5: Calculate the Euclidian Distance from the Ideal Worst

$$S_i^{+} = \left[\sum_j^m \left(V_{ij} - V_{ij}^{-}\right)^2\right]^{0.5} \qquad \textbf{3-24}$$

Step 6: Calculate Performance Score

$$P_i = \frac{S_i^{-}}{S_i^{+} + S_i^{-}} \qquad \textbf{3-25}$$

Step 7: Giving Rank

- Then take the first rank as design optimal solution for the given weight.

The flow chart to perform the selection method TOPSIS is discussed below.

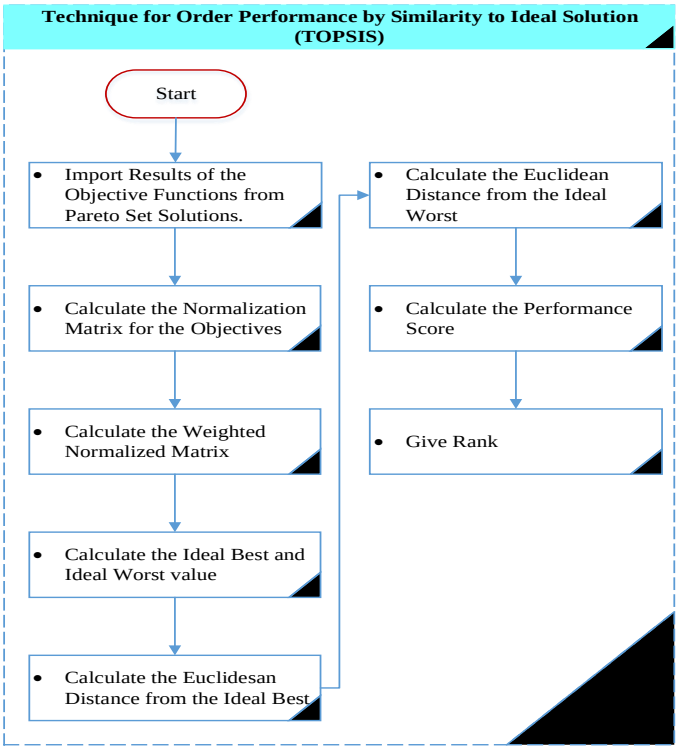


Figure 3-2 Flow Chart for TOPSIS Implementation

3.5 Verification

Design of a simply supported box girder satisfying all the flexural, shear limits is performed. after the design work is completed, the next step is to develop a MATLAB for Multi-objective optimization of the bridge structure, but before that, the optimization program is implemented on a simple cantilever welded beam and verified with previous works, as follow:

Design Optimization of a Welded Beam

Multi-objective optimization is presented here which to minimize the cost of the beam and also aims to minimize its maximum deflection subject to constants on shear stress, bending stress, and buckling load. there are four continuous design variables h,l,t, and that correspond to x1,x2,x3, and x4 and are shown in the figure below it is evident that minimization of the cost will lead to smaller dimensions of the beam. When the beam dimensions get smaller, the end deflections get bigger, and thus the conflicting objectives interact with each other to yield the set of non-dominated solutions.

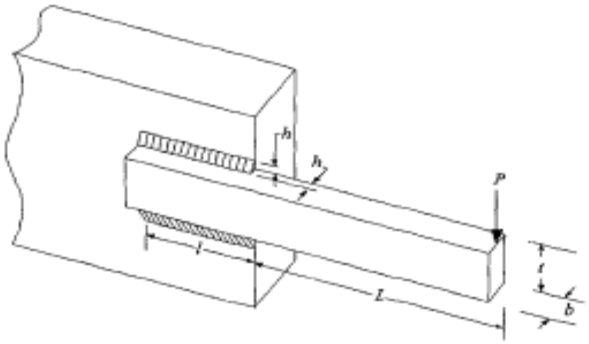


Figure 3-3 Welded Beam Design Problem (Hurst, 1998)

Minimize

$$f_1(x) = 1.1047 x_1^2 x_2 + 0.0481 x_1 x_4 (14.0 + x_2) \qquad 3\text{-}26$$
$$f_2(x) = \delta(x)$$

Subjected to

$$\tau(x) - \tau_{\max} \le 0 \quad , \quad \sigma(x) - \sigma_{\max} \le 0 \quad , \quad x_1 - x_4 \le 0 \quad , \quad P - P_c(x) \le 0 \quad ,$$

$$\tau(x) = \sqrt{(\tau')^2 + \frac{2\tau'\tau'' x_2}{2R} + (\tau'')^2} \quad , \quad \tau' = \frac{P}{\sqrt{2} x_1 x_2} \quad , \quad \tau'' = \frac{MR}{J} \quad , \quad M = P(L + \frac{x_2}{2}) \quad ,$$

$$R = \sqrt{\frac{x_2^{\,2}}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} \quad , \quad J = 2\left\{\frac{x_1 x_2}{\sqrt{2}}\left[\frac{x_2^{\,2}}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \quad , \quad \sigma(x) = \frac{6PL}{x_4 x_3^2} \quad , \quad \delta(x) = \frac{4PL^3}{x_4 x_3^2} \quad ,$$

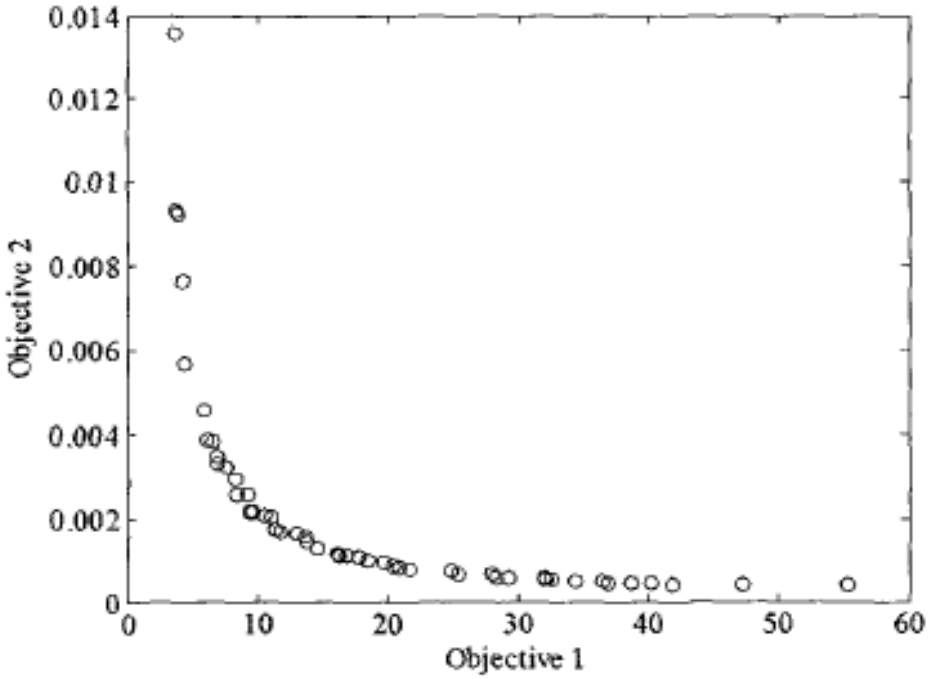
$$P_c(x) = \frac{4.013\sqrt{\frac{E G x_3^2 x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) .$$

Where $P = 6000lb$, $L = 14$, $\delta_{\max} = 0.25in$, $E = 30 * 10^6\ psi$, $G = 12 * 10^6\ psi$,

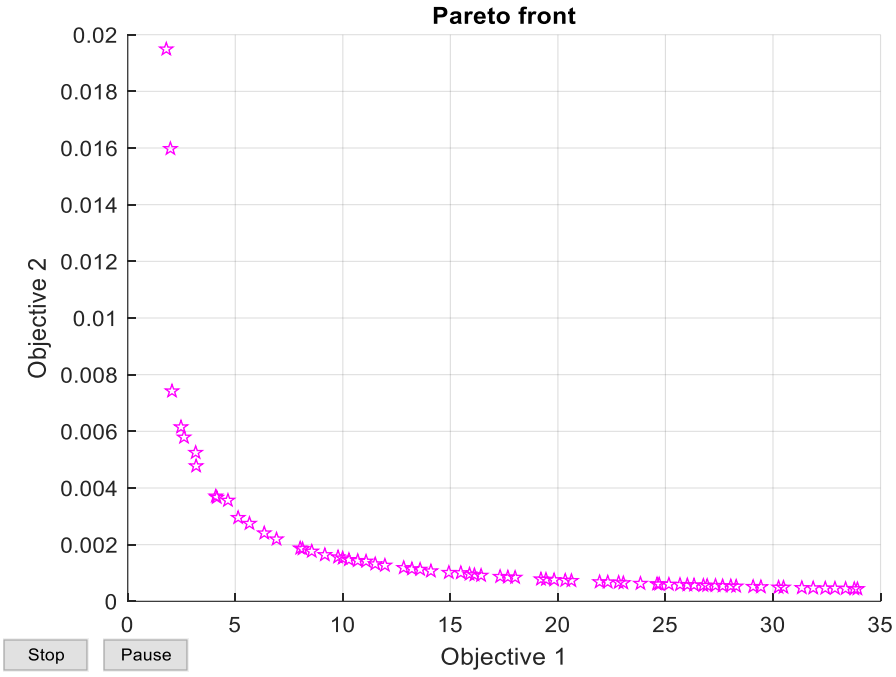
$\tau_{\max} = 13,600\,psi$, $\sigma_{\max} = 30,000\,psi$, $0.125 \le x_1 \le 10.0$, $0.1 \le x_2 \le 10.0$, $0.1 \le x_3 \le 10.0$

$0.125 \le x_4 \le 10.0$

Taking a result of the book (Hurst M.K, 1998) for the design of Cantilever beam having a population size of 100 has beam used in this study and the algorithm has been allowed to evaluate a maximum of 20,000 designs. minimum cost and the minimum end deflection design are presented below.



(a)



(b)

Figure 3-4 Pareto Front for Welded Beam (a) Results from (Hurst M.K, 1998) and (b) Result using Genetic Algorithm (gamultiobj).

The result obtained by using the developed code for this Study and results in Hurst are discussed below:

Table 3-1 Optimized Variables for Welded Beam (Hurst M.K, 1998)

Cost	Deflection	X1	X2	X3	X4
3.632894	0.013561	0.4532	3.71	7.0297	0.466
55.201977	0.000446	0.9366	6.5903	9.9965	4.9296

Table 3-2 Optimized variables and Objectives

Cost	Deflection	X1	X2	X3	X4
3.5555555	0.013561	0.1838	0.18132	9.9975	4.9983
54.201977	0.000446	0.2435	0.15261	9.97998	0.58038

The results are similar, so concluded that there is no problem with the programed MATLAB. So, the code can be implemented for the design of the Post-tensioned box girder bridge.

CHAPTER 4 PROBLEM FORMULATION AND RESULTS

4.1 General

An appropriate optimization problem has to be formulated for the present bridge structure to be solved in the multi-objective optimization method. In this chapter, the various components of the optimization problem of the present study are described. The various components for the design and analysis of the bridge system, an objective function, implicit constraints, and explicit constraints that are required are discussed. Code is developed using MATLAB toolbox to obtain the optimum solutions of cost as well as safety for the design of a post-tensioned pedestrian box girder bridge.

4.2 Optimization Input Parameters

4.2.1 Bridge design data

Girder length of 50m is used and a bridge width of 4500mm is used and wearing coarse of 50mm, a concrete grade of 50MPa and 7wire low relaxation strand is used; the procedure for the design of the prestressed bridge and a MATLAB code for the design of the bridge is stated in Appendix A. The results from the optimization are briefly described in detail in the next chapter of this paper.

4.2.2 Objective function

In this study, the objectives are cost minimization and maximizing safety for the present bridge system by taking into account only the cost of materials.

Cost of the bridge

In this study only the material cost of the bridge structure is taken as the cost of the bridge and calculated as follow:

$$C_T = C_c + C_{ps} + C_{os} \quad (4\text{-}1)$$

Where, C_c, C_{ps} and C_{os} are the cost of the girder concrete, prestressing concrete, and ordinary steel reinforcement respectively. And the costs of the individual components are calculated as:

$$C_c = UP_c V_c \quad (4\text{-}2)$$

$$C_{ps} = UP_{ps} W_{ps} \quad (4\text{-}3)$$

$$C_{os} = UP_{os}(W_{OSD} + W_{OSG}) \quad (4\text{-}4)$$

Where, UP_c ,UP_{ps} , and UP_{os} are the unit prices considering the only material cost of (i) the prestressed girder concrete,(ii) prestressing steel and ordinary steel respectively; V_c ,W_{ps} ,W_{OSD} , and W_{OSG} are the volume of the box girder concrete, weight of prestressing steel and ordinary steel in girder respectively.

Safety factor objective

In this study safety of the bridge is calculated through safety factors$\gamma(x)$ for the ultimate limit states taking into account the flexure, transverse flexure, and shear. And the minimum of the safety factors became the governing.

$$S(x) = min[\gamma(x)] \quad 4\text{-}5$$

4.2.3 Design variables and constraint design parameters

There are large numbers of parameters that control the design of the bridge such as cross-sectional dimensions of a girder, number of strands per tendon, number of tendons, girder reinforcement, anchorage system, prestress losses, concrete strength, etc. The design variables and variable types considered in the study are tabulated in table 4.1. A typical cross-section of the box–girder is illustrated in figure 4.1.

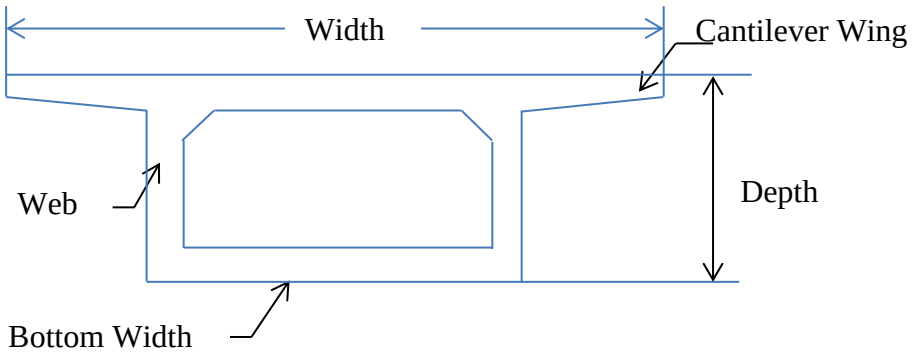


Figure 4-1Box Girder Bridge Layout

In the present study, the tendons arrangement is assumed as fixed and the tendons layout along the span is assumed as parabolic. The arrangement of tendons also depends on duct size and spacing, anchorage spacing, and anchorage edge distance.

Table 4-1 Design Variables

Design variables	Symbols
Girder depth (mm)	x_1
Top flange thickness (mm)	x_2
Top flange transition thickness (mm)	x_3
Bottom flange width (mm)	x_4
Bottom flange thickness (mm)	x_5
Length of overhang (mm)	x_6
Area of Shear reinforcement	x_7

4.2.3.1 Explicit constraints

These are specified limitations on the design variables which are derived from geometric requirements (superstructure depth, clearances, etc.), minimum practical dimension for construction, code restriction, etc. The constraint is designed as

$$X_L \leq X \leq X_U$$

Where, X=design variable, X_L =Lower limit of the design variable, X_U =Upper limit of the design variable. The design variables with explicit constraints considered in the study are tabulated in Table 4-2 below.

Table 4-2 Explicit Contraints

Design Variables	Explicit Constraint
Girder depth (x_1)(mm)	$1000 \leq x_1 \leq 3000$
Top flange thickness(x_2)(mm)	$170 \leq x_2 \leq 300$
Bottom flange width(x_3)(mm)	$300 \leq x_3 \leq 2000$
Bottom flange thickness(x_4)(mm)	$300 \leq x_4 \leq 600$
Length of overhang(x_5)(mm)	$530 \leq x_5 \leq 1800$
Web thickness (x_6)(mm)	$200 \leq x_6 \leq 500$
Area of Sear reinforcement (x_7) (mm)	$800 \leq x_7 \leq 1000$

4.2.3.2 Implicit constraints

These constraints represent the performance requirements or response of the bridge system. Total implicit constraints are considered according to the AASHTO standard specifications (AASHTO 2012). This constraint is formulated as below:

Flexural Working Stress Constraints

These constraints limit the working stresses in concrete and are given by:

$$f^L \le f_j \le f^u \quad \text{4-6}$$

$$f_j = \frac{f_j}{A} \pm \frac{f_j e_j}{S_j} \pm \frac{M_j}{S_j} \quad \text{4-7}$$

Where f^L =allowable compressive stress (Lower limit), f^u =allowable tensile stress (Upper limit), and f_j is the actual working stress in concrete; f_j , e_j , S_j , M_j =prestressing force, tendons eccentricity and moment at j[th] section respectively.

Allowable stresses for prestressed concrete

Compression stress:

- The stress limit due to the sum of the effective prestress, permanent loads are taken as $0.6 f'c$
- The stress limit in prestressed concrete at the service limit state after losses for a fully prestressed component in bridges other than segmentally constructed due to the sum of effective prestress and permanent load shall be taken as $0.45 f'c$
- The stress limit in prestressed concrete at the service limit state after losses for fully prestressed components in bridge other than segmentally constructed due to live load plus one-half the sum of the effective prestress and permanent load shall be taken as $0.40 f'c$

Tension stress:

The stress limit in prestressed concrete at the service limit state after losses for fully prestressed components in bridge other than segmentally constructed, which include bounded prestressing tendons and are subjected to not worth than moderate corrosion conditions shall be taken as follows: $0.5\sqrt{f'c}$ and tensile stress at service limit state before is taken as $0.63\sqrt{f'c}$.

In LRFD design, the total factored load is taken as follows:

$$Q = \eta \sum \gamma_i Q_i \qquad 4\text{-}8$$

Where:

η = a factor relating to ductility, redundancy, and operational importance (equal to 1.0 for this example)

γ_i =load factors

Q_i =specified loads

Check compressive stress in prestressed concrete component for service I

Service I: Q=1.00(DC+DW) +1.00(LL+IM)

Check tensile stress in prestressed concrete components for service III

Service III: Q=1.00(DC+DW) +0.8(LL+IM)

Check resistance for strength I

Strength I: Q=1.25DC+1.50DW+1.75(LL+IM)

For the above loading conditions, the allowable stress must be satisfied

$$f_{top} = \frac{f_j}{A} - \frac{f_j e_j}{S_j} + \frac{(M_{DL} + M_{DW})}{S_j} + \frac{M_{LL}}{S_j} \prec 0.6 * f'c \qquad 4\text{-}9$$

$$f_{bottom} = \frac{f_j}{A} + \frac{f_j e_j}{S_j} - \frac{(M_{DL} + M_{DW})}{S_j} - \frac{0.8(M_{LL+IM})}{S_j} \prec 0.5\sqrt{f'c} \qquad 4\text{-}10$$

Ultimate Flexural Strength Constraints

These constraints are according to ultimate strength design and are as follow:

$$M_u \leq \phi M_n \qquad 4\text{-}11$$

Where:

M_u = Maximum factored moment

ϕ = resistance factor

M_n = Nominal resistance

Maximum factored moment, M_u

The maximum moment for a simple span structure occurs at the midspan.

Strength I:

$$M_u = 1.25 * M_{DL} + 1.5 * M_{DW} + 1.75 * M_{LL+IM} \quad 4\text{-}12$$

Factored flexural resistance, M_r

The factored resistance M_r shall be taken as:

$$M_r = \phi M_n \quad 4\text{-}13$$

Assuming a tension–control section where the net tensile strain in extreme tension steel is ≥ 0.005 when concrete strain ε_c =0.003 and using ϕ=1.0, we have factored resistance:

Checking the assumption that the section is tension-controlled, $\varepsilon_t \succ 0.005$

$$\varepsilon_t = 0.003\left(\frac{d_t - c}{c}\right) \succ 0.005 \quad 4\text{-}14$$

$$\phi M_n = \phi\left[A_{ps} f_{ps}\left(d_p - \frac{a}{2}\right) + A_s f_y\left(d_p - \frac{a}{2}\right)\right] \quad 4\text{-}15$$

Where :

Aps = area of prestressing steel.

fps = average stress in prestressing steel at nominal bending resistance.

dp = distance from extreme compression fiber to the centroid of prestressing tendons.

As = area of non-prestressed tension reinforcement.

fs = stress in the mild steel tension reinforcement at nominal flexural resistance.

ds = distance from extreme compression fiber to the centroid of non-prestressed tensile reinforcement.

β1 = stress block factor specified in Article 5.7.2.2

a = cβ1; depth of the equivalent stress block.

Prestressing Steel Constraints:

Minimum prestressing steel constraints considered are as follow:

$$1.2M_{cr} \leq \phi M_n \quad 4\text{-}16$$

Where :

$$M_{cr} = \frac{I_{xx}}{c_1}\left(f_r + f_{pe} - f_d \right) \quad 4\text{-}17$$

$$f_r = 0.62\sqrt{f_c} \quad 4\text{-}18$$

$$f_{pe} = \frac{p_f}{A_g} + \frac{pe_c c_2}{I_{xx}} \quad 4\text{-}19$$

$$f_d = \frac{M_{dl} * c_2}{I} \quad 4\text{-}20$$

f_{pe} is compressive stress in concrete due to effective prestress forces

f_d is the stress due to un-factored self-weight.

Ultimate Shear Strength Constraints

AASHTO requires that for the strength limit state I

$$V_u \le \phi V_n \quad 4\text{-}21$$

Where :

$$V_u = \sum \eta \gamma_i V_i = 0.95\left[1.25\left(V_{DL} + 1.5V_{DW} + 1.75V_{LL+IM}\right)\right] \quad 4\text{-}22$$

ϕ = is shear resistance factor 0.9

V_n = nominal shear resistance which shall be the lesser of case (i) and (ii)

Case (i)

$$V_n = V_c + V_s + V_p \quad 4\text{-}23$$

$$V_p = F \sin\theta \quad 4\text{-}24$$

$$V_c = 0.0316\beta\sqrt{f_c'}\, b_v d_v \quad 4\text{-}25$$

$$V_s = \frac{A_v f_y d_v \left(\cot\theta + \cot\alpha\right)\sin\alpha}{s} \quad 4\text{-}26$$

Case (ii)

$$V_n = 0.25 * f_c b_v d_v + V_p \quad 4\text{-}27$$

$$V_p = F \sin\theta \quad 4\text{-}28$$

Requirement for shear reinforcement

Check if

$$V_u \geq 0.5\phi\left(V_c + V_p\right) \qquad 4\text{-}29$$

Check maximum spacing of shear reinforcement

Shear stress, v_u

$$v_u = \frac{V_u - \phi V_p}{\phi b_v d_v} \qquad 4\text{-}30$$

Maximum spacing of shear reinforcement is

$$v_u \prec 0.125 f_c' \qquad 4\text{-}31$$

4.2.3.3 Deflection Constraints

The deflection limits according to AASHTO LRFD guide specification for pedestrian bridge section 5, deflection of the bridge due to un-factored pedestrian live loading shall not exceed $\frac{1}{360}$ of the span length.

Deflection at mid-span due to initial prestress (for parabolic tendon profile) is computed as:

$$\Delta = \frac{L^2}{8E_c I}\left(M_e + \frac{5}{6}M_c\right) \qquad 4\text{-}32$$

Where: M_e is the primary moment in the end, $P_j e_{end}$, and M_c is the primary moment at mid-span $P_j e_c$ .un-cracked gross section properties are often used in calculating camber. For deflection at service loads, cracked section properties, that is, a moment of inertia I_{cr} should be used at the post-cracking service–load stage.

Deflection due to girder self-weight at mid-span is computed as :

$$\Delta_g = \frac{5w_g L^4}{384E_{ci}(I)} \qquad 4\text{-}33$$

Deflection due to live load :

$$\Delta_{LL} = \frac{5*W_{LL}*L^4}{384E_{ci}(I)} \qquad 4\text{-}34$$

The live load deflection constraint is as follow:

$$\Delta_{LL} \prec \frac{L}{360} \qquad 4\text{-}35$$

4.2.4 Constant design parameters

The following parameters are constant throughout this study for the design of the post-tensioned box-girder pedestrian bridge.

- concrete strength - the concrete strength used in this study is f_c=50MPa
- Tensile Strength – the concrete direct tensile strength is $f_r = 0.63 * \sqrt{f_c}$, the tensile strength for the prestressing strand is taken as f_{py}= 1860 MPa, and the tensile strength of the steel is 410MPa.
- Modules of elasticity – the module of elasticity for prestressing strand is taken 197,000 MPa. And modulus of elasticity for the concrete structure is $E_c = 0.043 K_1 \gamma_c^{1.5} \sqrt{f_c}$.

4.3 Performing Multi-Objective Optimization using Genetic algorithm

This section discusses how to perform a multi-objective optimization of post-tensioned pedestrian bridges using the multi-objective genetic algorithm in the Global Optimization Toolbox. This bridge design, having several variables. It is planned to minimize one objective and maximize the other, each having several decision variables. The design framework used for the design optimization is discussed below.

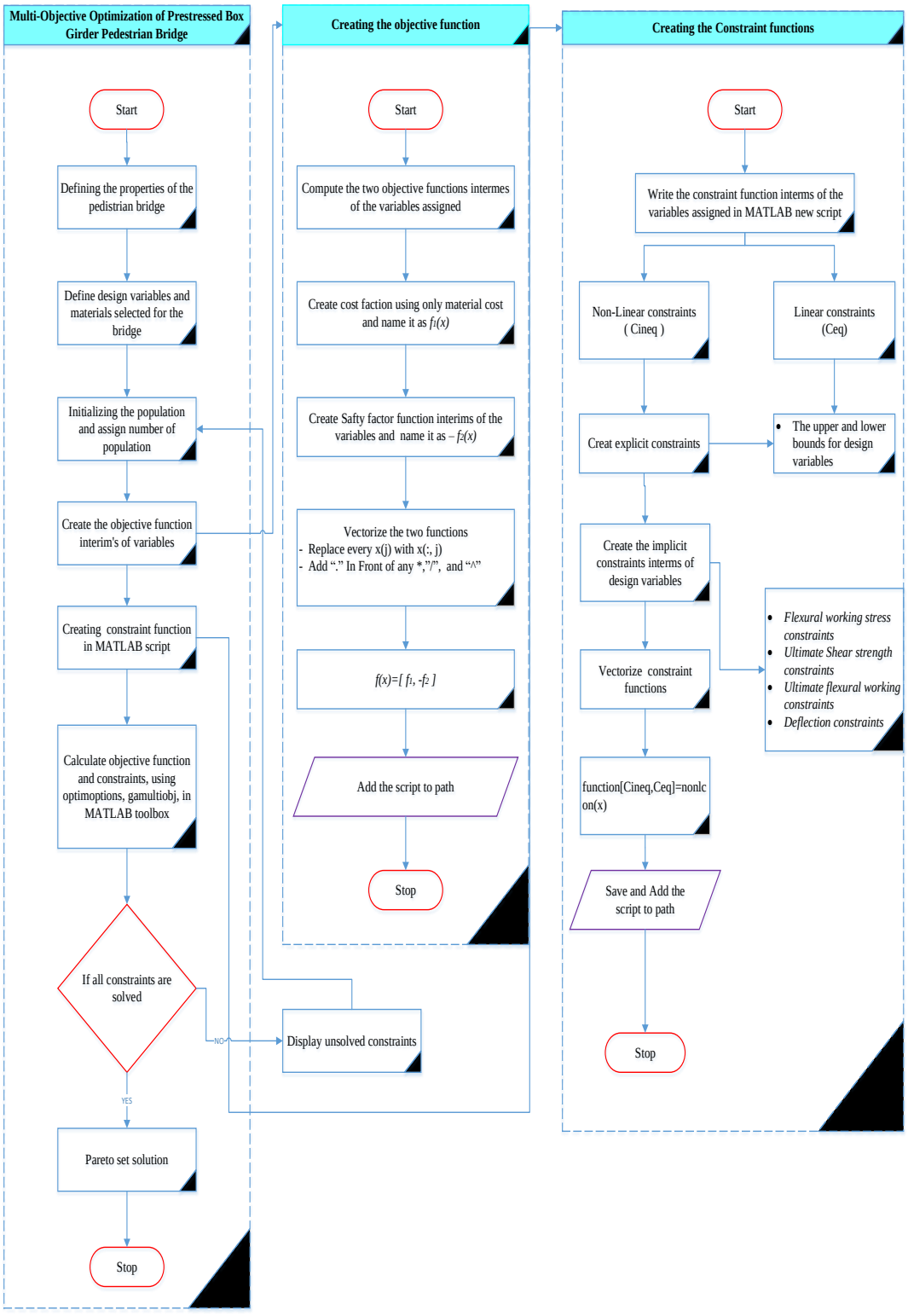


Figure 4-2 Flowchart for Multi-objective Optimization of Post-tensioned Pedestrian Bridge

The procedures used to develop the design framework for the box girder bridge are discussed below: -

Step 1: - Initializing

The first step is creating an initial population. The algorithm creates the population, or you can give an initial population or a partial initial population by using the Initial Population Matrix option. by default, the algorithm creates a population that is feasible to bounds and linear constraints but is not necessarily feasible to nonlinear constraints. So, in this study, we use the default random population created by the algorithm. A population size of 1000 was taken and the code is described in Appendix B.

Step 2: - Compute Objective function

To use Optimization Toolbox functions, first, write a file (or an anonymous function) that computes the function you want to optimize. This is called an objective function for most solvers, or a fitness function for genetic algorithms. The function should accept a vector, whose length is the number of independent variables, and return a scalar. For vectorized solvers, the function should accept a matrix, where each row represents one input vector, and return a vector of the objective function value.

Minimizing and Maximizing the Objective

Global optimization toolbox optimization functions minimize the objective or fitness function. That is, they solve problems of the form:

$$\min_x f(x)$$

To maximize the function, minimize the point at which minimum occurs which is the same as the point at which the maximum occurs.

Write an Objective Function File

Two Objective functions which are the cost of the box girder bridge and safety of the structure are needed to be created so to write this function the following procedure is used:

- Select New from the MATLAB file menu. A new file opens in the editor
- Write all objective functions in terms of a variable.
- Save the file in a folder on the MATLAB path

Step 3: - Write a Vectorized Objective Function

The solver optionally computes the objective functions of a collection of vectors in one function call. This method can take less time than computing the objective functions of the vectors serially.

To vectorize a function the following two methods are implemented: -

- Replace every $X(j)$ with $X(:,j)$
- Add "." In any "/", "*" and "^"

Step 4: - Write Constraints

Optimization toolbox functions accept bounds, linear constraints, or nonlinear constraints. It is important to set bounds. For nonlinear constraints took every equation in less than zero forms and express it interims of variables.

Step 5: - Vectorized Constraints

The solver optionally computes the nonlinear constraint functions of a collection of vectors in one function call. This method can take less time than computing the objective function of the vectors serially. This method is called a vectorized function call. The vectorized nonlinear constraint function for this study is presented below in the new MATLAB subscript and this file is saved in the MATLAB path.

The vectorized nonlinear constraint function for the post-tensioned box girder is written in Appendix B.

Step 6: - Set and Change Options

For all Optimization Toolbox solvers except Global Search. The recommended way to set options is to use the Optimoptions function. Set Global Search options using their name-value pairs.

Step 7: - Create the main function

Creating a new script in the MATLAB named main_.m function for the optimization of the bridge structure to get the output Pareto front, and this function is written in Appendix B.

4.4 Results and Discussion

Figure in the below shows the Pareto set for the two objective functions. All the points in the pareto set are optimal solutions for the design considering the two objectives whose objective values cannot be improved without worsening the value of the other.

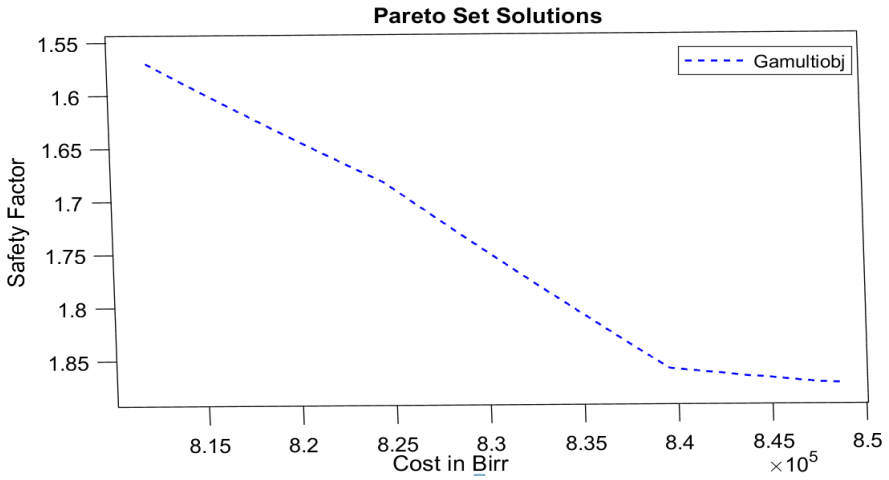


Figure 4-3 Pareto Set of Solutions

The result shows that the two objectives are conflicting objectives reducing the material consumption decreases the cost of the structure as well as the safety of the structure. As we can see in figure 4.3, firstly the safety factor of the bridge increase with the cost of the bridge having a linear relationship $S = 1e - 05C - 6.5172$ and regression coefficient $R^2 = 0.9822$. This means to increase a safety factor by 10% results in a cost increment of 3%.

Traditional design methods for bridge design involve a trial-and-error procedure. The geometrical layout is a priori defined to perform the structural analysis. The stress and displacement are compared with the allowable values, according to the code specified, otherwise, the remaining variables of the reinforcing steel are calculated based on the code's ultimate limit state. The final design does not guarantee to be optimal.

Unlike traditional design, optimization search for the value of design variables that minimize the objectives as well as that satisfies limit state constraints. In this study shear, flexure, and transverse flexure is the limit states taken into account and regarding the factor of safety, the shear strength limit state is the one governing in the design and safety consideration, so below the effect of the variables the depth and area of shear reinforcement are discussed below.

Concerning the depth (D), the minimum value is 2302 mm and the range varies. corresponding to the relationship b/n the depth and cost of the bridge Figure 4-4 below illustrate that increasing the depth of the bridge shows an increase in the cost of the bridge, having linear relation with cost, means increasing the depth by 10% results in 3% increase in the cost of the bridge structure), until the cost of the bridge is 8.3*10^5 birr but after this, the depth of the bridge became constant as the cost of the function increase. The depth of the structure again is directly related to safety as shown in figure 4.5, as the depth of the structure increases the safety also starts increasing, having linear relation $D = 1674.3S - 297.28$ with a regression coefficient of $R^2 = 0.9874$. This means that increasing the depth by 10% results in a safety increment of 71.76%

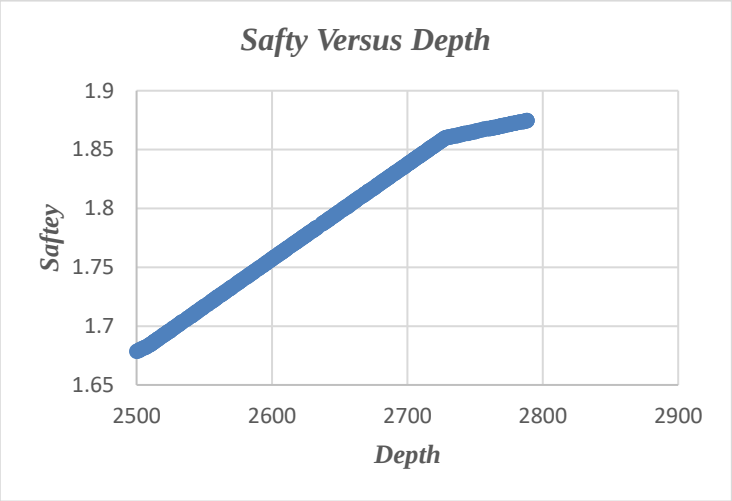


Figure 4-4 Depth Versus Safety of the bridge

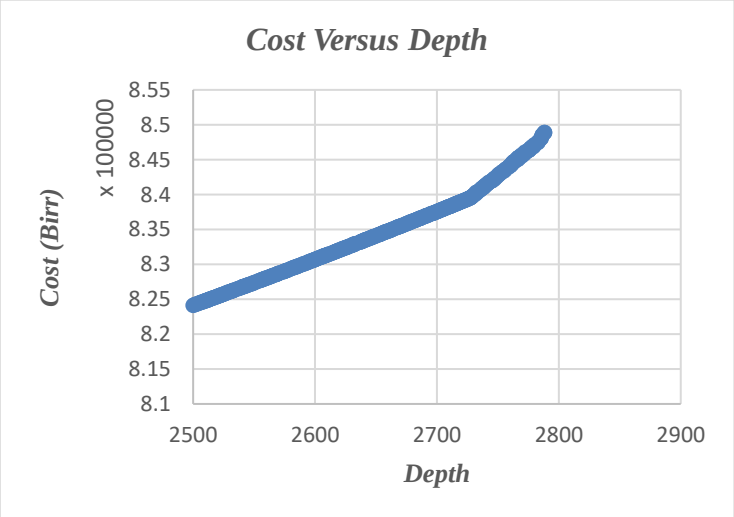


Figure 4-5 Depth Versus Cost of the bridge

Fig. 4-6 shows the relation between shear reinforcement and safety factor, increasing the area of reinforcement has less impact on the safety of the structure until it reaches a safety factor of 1.85, after this point increasing the area of the shear reinforcement has a great impact on the safety of the bridge and fig. 4-7 illustrates that increasing the area of the shear reinforcement has a great impact on the cost of the structure as seen in the figure below.

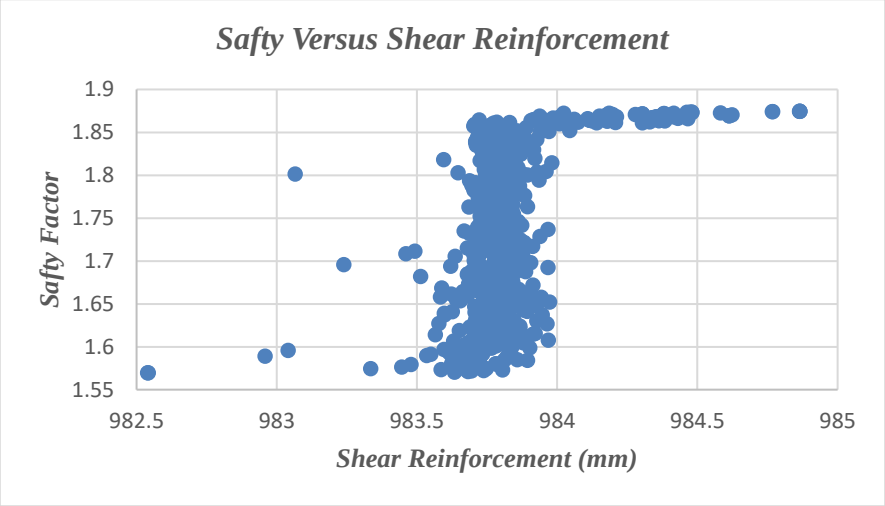


Figure 4-6 Safety Versus Transversal Reinforcement

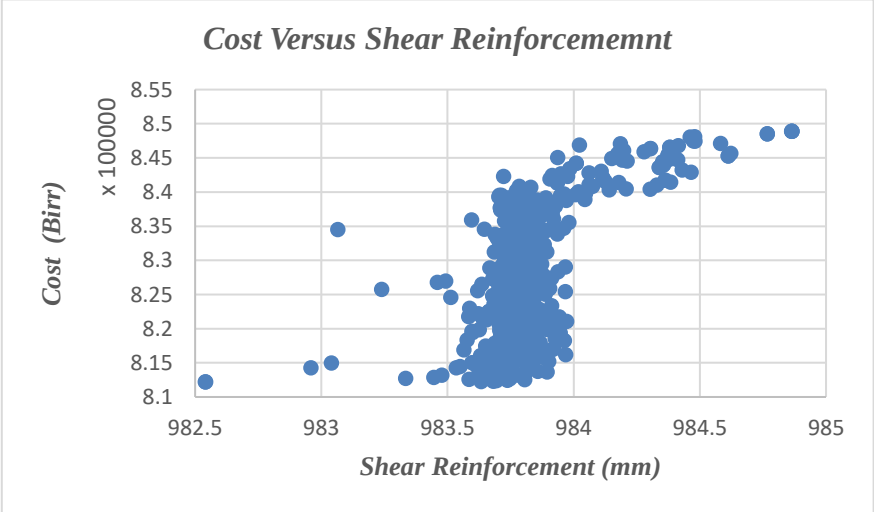


Figure 4-7 Cost Versus Shear Reinforcement

Finally, a result of 639 solutions was taken from the Pareto front for both objective functions of cost and safety, by giving a weight of 75% and 25% for the cost and safety factor function respectively. Then decision-making method TOPSIS is used to choose the optimal solution, and a cost of 8.4*10^5 birr and Safety of 1.87 is taken as the optimal solution for the specific design.

CHAPTER 5 CONCLUSIONS AND RECOMMENDATIONS

5.1 Conclusion

The study has presented a design framework for conducting a multi-objective optimization design of a post-tensioned box-girder pedestrian bridge. The design is performed for optimization of box-girder pedestrian bridge considering only material cost, and safety of the bridge for the ultimate loading satisfying all limit states required in ASSHTO. Multi-objective optimization methodology is developed using MATLAB toolbox algorithm, gamultiobj, and Pareto optimum solutions are obtained for the two objective functions. The following conclusions are reached:

- The design of the bridge and the optimization technique is demonstrated. the applied MATLAB routine *gamultiobj* algorithm provides improved designs and many optimal solutions, with freedom for the designer of selecting preferred solutions, which leads to cost-effective and safe design results.
- Proved that multi-objective optimization is practically feasible and more informative and beneficial for design optimization of post-tensioned pedestrian bridges.
- A genetic algorithm is proved to be the best optimization technique to solve the multi-objective optimization of a post-tensioned pedestrian bridge. Considering all linear and nonlinear constraints and design variables.
- The present study formulation and solution method demonstrate the interaction of the objective functions with some of the design variables for simply supported post tensioned pedestrian bridge. So that better design decisions can be made when the designer is aware of the interaction between the two competing objectives.

design examples are performed to demonstrate the proposed design outline, and the following conclusions are made: -

- Based on the results obtained it is concluded that the cost and safety of the bridge structures have a linear relationship $S = 1e-05C-6.5172$, where S and C represent safety factor and cost respectively. So that increasing the safety factor by 10% will linearly increase the cost of the structure by 3%, in addition when came to the design variables increasing the depth by 10% results 3% increase in the cost of the bridge structure and a safety factor increment of 71.76%, the other design variable is a shear area of reinforcement, which shows that increasing the area of the shear reinforcement by 2% make the certain increase in cost and the safety factor increment of 65%.
- Then decision-making method TOPSIS is used to choose the optimal solution and is concluded that cost of 8.4*10^5 birr and safety factor of 1.87 is taken as the optimal solutions for the specific design.

5.2 Recommendations

This thesis provides a design optimization outline using MATLAB routine *gamultiobj* algorithm for simply supported post-tensioned box girder pedestrian bridge design optimization. It is recommended for the designers to use this outline for the optimal design of a post-tensioned pedestrian bridge which provides multi-solutions. In addition, a designer is recommended to use TOPSIS for selecting a single optimal solution from the sets of solutions by giving weights to the design Parameters.

It is recommended that this study can be improved and extended far beyond what is implemented. Such as: -

- Further study in multi-objective optimization using an engineering Software CSI Bridge with MATLAB using API interface can be performed to make the process easy for future implementation.
- Further study on multi-objective optimization and decision-making methods, such as AHP (analytic hierarchy process), can be study in the future… etc.

REFERENCES

AASHTO. AASHTO LRFD bridge design specifications (2012). Washington, DC, US.

Alqedra M.. Araf M.,and Ismail M. (2011). Optimum Cost of Prestressed and Reinforced Beams using Genetic Algorithm. Journal of artificial intelligence, 4(1):76-88.

Aydın Z. and Ayvaz Y. (2010). Optimum topology and shape design of prestressed concrete bridge girder using genetic algorithm. Journal of structural and multidisciplinary optimization, 41:151–162.

Barakat S., Bani-Hani K., and Tana M.Q. (2004). Multi-objective reliability-based optimization of prestressed concrete beams. Journal of Structural Safety, 311-342.

Bhinge D. and Fernandes .R.J. (2018). Optimization of I-girder bridge using genetic algorithm method. International Journal of scientific research, ISSN No 2277 - 8179.

Camp,C., pezeshk, S., and Cao, G. (1998). Optimization of Two-Dimensional Structures Using a Genetic Algorithm. Journal of Structural Engineering, Vol.1, 551-559.

Coelio A.C. , Lamont B.G. (2004). Application of Multi-Objective Evolutionary Algorithm (Vol. 1). World Scientific Publishing Co. Pte. Ltd. ISBN 981-256-106-4.

Deb Kalyanmoy. (2001). Multi-objective optimization using evolutionary algorithm(1st ed.) , New York, ISBN 0-471-87339-X.

European Standard. (1992). Eurocode 2: Design of Concrete Structures. European Commitee for Standardization.

Lamont B.G. and Coello C.A. (2004). Application of Multi-objective evolutionary algorithms. Advance in natural computation, Vol.1, ISBN 981-256-106-4.

Hurst M.K. (1998). Prestressed Concrete Design (2nd Edition ed.). New York.

Morab A.N. and Fernandes R.J.(2018). Optimization of Box Girder Bridge Using Genetic Algorithm Method. IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE), e-ISSN: 2278-1684.p-ISSN: 2320-334X.

Konak A., Colit W.D., and Smith E.A. (2006). Multi-objective optimization using genetic algorithm. Reliability Engineering and System Safety , 992–1007.

Krishenan Raju.N. (2010). Prestressed concrete (2nd ed.). Tata Mcc Grahil.

Kumar A. and Kaur M. (2017). Cost Optimization of Beam by Genetic Algorithm. International Journal of Advance Reseach in science, ISSN:2319-9354.

Lounis Z. and Cohn M. Z. . (1993). Multi-objective Optimization of prestressed concrete structures. Journal of Structural Engineering, ASCE, 119(3): 794-808.

Marti V., Vidosa G.,and Alcala J. (2009). Heuristic optimization of prestressed concrete precast pedestrian bridges. Computer Aided OPtimum Design in Engineering, XI 121.

MathWork. (2018). Optimization Toolbox User's Guide. MathWork.

Rama K. S., Sudahir R., Kumar S., and Vickranth V. (2010). Study and Behaviour of Box Girder Bridge. research gate.

Segura G., Yepes V., Alcala J. and Lopez P. (2015). Hybrid harmony search for sustainable design of post-tensioned concrete box-girder pedestrian bridge . Engineering Structures, 112-122.

T.Y. Lin and Ned H.Burns. (1981). Design of Prestressed Concrete Structure. 3rd edition, New York: John Wiley & Sons, Inc.

Wai-Fah Chen and Lian Duan.(2014). Bridge Engineering Handbook, SUPERSTRUCTURE DESIGN, CRC press Taylor and Francis Group.

Yogyakarta, and Yoyong. (2015). cross section and prestressing force optimization of prestressed bridge. ResearchGate.

Young- Jou Lai, Ting-Yun Liu, Ching-Lai-Hwang. (1994). TOPSIS for Multiple objective decision making. Science Direct, Volum 76, issue3.

APPENDIX: A POST-TENSIONED BOX GIRDER BRIDGE DESIGN

MATLAB design code for post-tensioned box girder pedestrian bridge.

Step- 1 Determining cross-sectional geometry

Algorithm A-0-1 Determining the Cross Sectional Geometry

```
%structural depth D, the span of the bridge(L), width of the girder(W),the
overhanging width(Lc),bottom flange width(bbot)
%Thickness of the web(tf) , thickness of the top flange(tf), thickness of the
bottom flange (tbf), the fillet height(s) ,moment of inertia of the section
(Ixx),
%Area of the sections(Ai),total area of the box structure (At) ,centroid of the
areas (Yi), centroid distance from the top fiber (Yii)
%Centroid of the box structure (Yt),volume of the concrete (V),the effective
flange width(S)
%Unite weight of the normal concrete in kg/m^3(rc),gravity(g) ,width of the
barrier(Wb)
%h2=height of the girder form the bottom of the slab to the bottom of the
girder
L=50000; D=3000; W=3500; Lc=800; tw=300; bbot=W-2*(Lc)-tw; S=W-2*Lc ;
%AASHTO 9.7.1.1 the concrete depth should not be less than 175mm
tf=180;
if tf> S/6, tf = S/6 ;
disp('if')
elseif S/6>tf> S/18,tf = tf;
disp('elseif')
end
s=100; A1=W*tf; Y1=0.5*tf; Y11=0.5*tf; A2=0.5*s*s; Y2=(s)/3; Y22=tf+Y2; h2=(D-
tf);
A3=h2*tw; Y3=h2/2; Y33=tf+Y3; tbf=300; A4=bbot*tbf; Y4=0.5*tbf; Y44=D-
Y4;A5=0.5*s*(Lc-tw); Y5=s/3; Y55=tf+Y5;
Ytot=(A1*Y11+2*A2*Y22+2*A3*Y33+A4*Y44+2*A5*Y55)/(A1+2*A2+2*A3+A4+2*A5);
Ix1=(W*tf^3)/12;
Ix2=s^4/(12);
Ix4=((tbf^3)*(bbot))/12;
Ix3=(h2*tw^3)/12;
Ix5=(A5*s^2)/6;
Ixx=(Ix1+A1*(Y11-Ytot)^2)+2*(Ix2+A2*(Y22-Ytot)^2)+2*(Ix3+A3*(Y33-
Ytot)^2)+(Ix4+A4*(Y44-Ytot)^2)+2*(Ix5+A5*(Y55-Ytot)^2)
At=A1+2*A2+2*A3+A4+2*A5;
V=At*L;
```

Step- 2 Material Selection

Algorithm A-0-2 Material Selection

```
%AASHTO 2.4 concrete strength should be a minimum of 40Mpa
fc=50;
%Tensile strength , for most concretes the direct tensile strength
fr=0.62*fc^0.5 ,
%for prestressing strands the tensile strength is fpu= 1860MPA
fr=0.62*(fc^0.5);
fpu=1860;
fy=420;
%yield strength for strand is given by 0.9*fpu for low relaxation strand
fpy=0.9*fpu;
%modules of elasticity for steel reinforcement shall be assumed
%Es=200,000MPa , for pre-stressing strand Ep=197,000MPa
Es=200000;
Ep=197000;
%modules of elasticity of concrete
```

```
Ec=4800*(fc^0.5);
%unit density of a concrete(rc) ,refer AASHTO table 3.5.1.1 in kg/mm3
rc=2320*10^-9;
g=10;
```

Step- 3 Load calculation and prestressing force

Algorithm A-0-3 Load Calculation and Prestressing Force

```
%%%%%%%step 3 dead load due to the self-weight of concrete (PDL1)
PDL1=At*rc*g;
%Barrier rail weight acting at service stage after all loos is 11.5N/mm
Pb=11.5;
%thickness of the wearing surface of 75mm
twer=75;
% Distributed load of the wearing surface having a density of 2250kg/m3
rw=2250*10^-9;
Wb=200;
DW=(W-2*Wb)*twer*rw*g;
%total dead load due to un-factored self-weight, barrier, and wearing weight
PDL=PDL1+DW+Pb;
%determination of the live load LL and dynamic load allowance IM
%AASHTO-3.6.2.1 state that dynamic load allowance shall not be applied to
%pedestrian loads or IM=1.0 should be used
%and for bridge only for pedestrian and bicycle traffic shall be designed
%for a live load of 4.1*10^3 pa
PLL=18.45;
% for the pedestrian load we will not consider IM
%load combination
% service I check compressive stress in prestressed concrete component for
Q1=1.00*(PDL)+1.00*(PLL);
%service II check tensile stress in prestressed concrete component
Q2=1.00*(PDL)+0.80*(PLL);
%check resistance for strength I
Q3=1.25*(PDL)+1.5*DW+1.75*(PLL);
%%%%%%% Determining the prestressing force
% the efficiency of the cross-section p
P=Ixx/(At*Ytot*(D-Ytot));
%permissible concrete stress (fa) and permissible tensile stress (fta)
fa=0.6*fc;
fta=0.5*fc^0.5;
%Moment required to produce the permissible compressive stress at the bottom-
most fiber of the girder
c1=Ytot;
c2=D-Ytot;
Mbc=(fa*Ixx)/c2;
%Moment required to produce the permissible tensile stress at the bottom-most
fiber of the girder
Mbt=-(fta*Ixx)/c2;
%Moment required to produce the permissible compressive stress at the topmost
fiber of the girder
Mtc=(fa*Ixx)/c1;
%Moment required to produce the permissible tension stress at the topmost fiber
of the girder
Mtt=-(fta*Ixx)/c1;
%The mid-span bending moment due to the applied load at service limit state I
M=(Q2*L^2)/8;
%moment due to the own weight at mid-span
MD=((PDL1+Pb)*L^2)/8;
%moment due to the wearing surface at mid-span
Mw=((DW)*L^2)/8;
%moment due to the live load  at mid-span
ML=((PLL)*L^2)/8;
% the center of gravity o the strand at mid-span is assumed to be located
% at 5% of the girder height
ybs=D*0.05;
e=500;
```

```
%minimum prestressing force required to limit the bottom girder tension
Fbmin=(M+Mbt)/(P*c1+e);
%Max prestressing force required to limit the bottom girder tension
Fbmax=(M+Mbc)/(P*c1+e);
%Max prestressing force required to not exceed the minimum tension in top
girder tension
Ftmax=(M-Mtt)/(e-P*c2);
%Min prestressing force required to not exceed the minimum compression in top
girder tension
Ftmin=(M-Mtc)/(e-P*c2);
%the preliminary prestressing force is usually determined based on the service
limit state III load condition at mid-span
F=(M+Mbt)/(P*c1+e);
%limits of eccentricity with regards to top of the girder
emaxt=((M-Mtt)/F)+P*c2;
emint=((M-Mtc)/F)+P*c2;
%limits of eccentricity with regards to bottom of the girder
emaxb=((M+Mbc)/F)-P*c1;
eminb=((M+Mbt)/F)-P*c1;
%Area of the strand (Ast)having a tensile strength of 1860 Mpa and
15.24diameter and
%Area required for the given value of prestressing force (Aps), number of
strands required (N)
Ast=((15.24)^2*pi)/4;
Aps=F/fpy;
N=Aps/(7*Ast);
```

Step- 4 Calculate prestressing Losses

Algorithm A-0-4 Prestress Loss Calculation

```
%Friction loss fpf , AASHTO art.5.9.5.2.2b
%ep is the distance b/n two centroid points Lp is the horizontal distance
%b/n two control points , K is the web friction coefficient 0.0002 ,
%U is the coefficient of friction 0.25
fpj=0.9*fpu;
K=0.0002;
U=0.25;
ep=e;
Lp=0.5*L;
a=2*ep/Lp;
x=780;
fpf=fpj*(1-(exp(-(K*x+U*a))));
%2 Anchorage set loss fpa
%for an anchor set thickness of L1=10mm and E=200,000mpa ,Lpf=50m
Lpf=50000;
L1=10;
Lpa=(Ec*L1*Lpf/fpf)^0.5;
if Lpa>Lpf
    fpa=0; %no anchorage set loss
    elseif Lpa<Lpf
    f=2*fpf*Lpa/Lpf;
    %assuming horizontal distance X=Lpa
    fpa=f;
end
%3 elastic shortening loss fpes AASHTO 5.9.5.3.b
%fcgp=(F/A)+(F*e^2/Ixx)+(M*e/Ixx) , where M is the moment due to own weight,
%fcgp is the sum of the concrete stress at the center of gravity of the
%prestressing force after jacking for post tensioned member, for simply
%supported structure calculated at the center section of the span
fcgp=(F/At)+(F*e^2/Ixx)+((MD+Mw)*e/Ixx);
fpes=(N-1)*Ep*fcgp/(2*N*Ec);
%4 time dependent loss
%AASHTO provides s table to estimate the accumulated effect of time dependent
%losses resulting from the creep and shrinkage of concrete and relaxation of
%the steel tendons .there for fptm=145MPa
```

```
fptm=145;
fpt=fpf+fpa+fpes+fptm;
% To determine the jacking force Pj, after loss the initial prestress force
% coefficient Fpci and final prestress coefficient Fpcf
Fpci=1-(fpf+fpa+fpes)/fpj;
Fpcf=1-(fpt/fpj);
%the required prestressing force at transfer (before any loss)
Pi=F/Fpcf;
Aps1=Pi/fpy;
N2=Aps1/(7*Ast);
```

Step- 5 Check prestressing and Concrete strength limit

Algorithm A-0-5 Checking Stress Limits

```
%%check prestressing stress limit at service limit state
%the initial stress in prestressing steel before transfer <=0.75*fpu
fpbt=0.75*fpu;
fpe=fpbt-fpt;
if fpe<0.8*fpy
    fprintf('prestressing stress limit is satisfied');
end
%Stress at the top of CIP box girder (ftop) due to the own weight should be <fa
=0.6*fc Mpa
ftop=(MD+Mw)*c1/Ixx;
if ftop<fa
    fprintf('compressive strength limit is satisfied');
disp('if')
elseif ftop >fa
    fprintf('compressive strength requirement is not satisfied');
disp('elseif')
end
% Check concrete strength using service load I at bottom of the girder
M1=Q1*L^2/8;
fcsb=Fpcf*Pi/At +Fpcf*Pi*e*c2/Ixx -M1*c2/Ixx;
if fcsb<0.45*fc
    fprintf('compressive strength limit is satisfied')
disp('if')
elseif fcsb >0.45*fc
    fprintf('compressive strength limit is not satisfied')
disp('elseif')
end
% Check tensile stresses at bottom of the girder under service II
% fbot<fta
M2=Q2*L^2/8;
fbot=(Pi*Fpcf/At)+(Fpcf*Pi*e*c2/Ixx)-(M2*c2/Ixx);
if fbot<0.5*((fc)^0.5), Pi= Pi;
disp('if')
elseif fbot>0.5*((fc)^0.5)
    fprintf('you have to change the prestressing force')
disp('elseif')
end
```

Step -6 Design for strength limit state – flexure

Algorithm A-0-6 Strength Limit State - Flexure

```
%Maximum factored moment Mu
Mu=1.25*MD+1.5*Mw+1.75*ML;
%Average prestressing steel stress
fpy=0.9*fpu;
K=2*(1.04-(fpy/fpu));
B1=0.85-((0.05)*(fc-28)/7);
if B1<0.65, B1= 0.65;
disp('if')
elseif B1>0.65 ,B1=B1;
```

```
disp('elseif')
end
%%assume that the compressive area is rectangular and fs=fy the distance from
neutral axis to extreme compressive fiber is as follows
fs=fy;
dp=D-0.05*D;
cc=25;
d=16;
ds=D-cc-d/2;
As=(0.85*fc*W*D-((0.85*fc*W*D)^2 -1.7*fc*W*Mu)^0.5)/fy;
Asmin=0.002*At;
if As < Asmin, As= Asmin;
disp('if')
elseif As >Asmin ,As=As;
disp('elseif')
end
C=(Aps*fpu+As*fs)/(0.85*fc*B1*W+K*Aps*(fpu/dp));
if C>tf ,C=C;
elseif C<tf ,C=tf;
end
a=B1*C;

fps=fpu*(1-(K*C/dp));
if fps<0.5*fpu, fs= fy;
disp('if')
elseif fps >0.5*fpu
    fprintf('the equation is correct according to AASHTO5.7.3.1.1')
disp('elseif')
end
de=(Aps*fps*dp+As*fy*ds)/(Aps*fps+As*fy);
if C/de<=0.42
    fprintf('the reinforced concrete section is not over reinforced ')
elseif C/de>0.42
    fprintf('the RC section is over reinforced')
end
%%factored moment resistance
Mn=(Aps*fps*(dp-0.5*a)+As*fy*(dp-0.5*a));
%%check the assumption that the section is tension controlled.
%%et>0.005
dt=D-0.05*D;
et=0.0030*(dt-C)/C;
if et > 0.005
    fprintf('tension controlled')
disp('if')
elseif et < 0.005
    fprintf('tension is not controlled')
disp('elseif')
end
```

```
%%%% minimum reinforcement
%%the amount of prestressed tensile reinforcement at any section requires that
%%flexural resistance Mr, equals the lesser of i,1.33*Mu and Mcr
fr=0.62*fc^0.5;
fpe=(Pi/At)+(F*e*c2/Ixx);
fd=(MD+Mw)*c2/Ixx;
Mcr=Ixx*(fr+fpe-fd)/Ytot;
Mr=min(Mcr,1.33*Mu);
if Mn > Mr
    fprintf('the minimum requirement is satisfied')
disp('if')
elseif Mn < Mr
    fprintf('minimum requirement is not satisfied so you have to increase Ast')
disp('elseif')
end
```

Step- 7 Design for Shear- Strength

Algorithm A-0-7 Strength Limit State - Shear

```
%%%% step-7 design for shear-strength limit state I
Vu=0.95*Q3*L;
%critical section for shear design
%the effective depth for shear
dva=[(de-0.5*a) (0.9*de) (0.72*D)];
dv=min(dva);
O=atan((e/(0.5*L)));
Vp=Pi*sind(O*180/pi);
if Mu> abs((Vu-Vp)*dv)
    fprintf('ok')
end
%%Contribution of concrete to nominal shear resistance
fpo=0.7*fpu;
Nu=0
ex=(((Mu/dv)+0.5*Nu+abs(Vu-Vp)-Aps*fpo))/(Es*As+Ep*Aps1)
es=ex;
B=4.8/(1+750*ex)
tt=29+3500*es %%AASHTO 5.8.3.4.2.3
%%concrete contribution for shear
bv=0.2*tw
Vc=0.083*B*(fc)^0.5*bv*dv
%% Requirement for Shear Reinforcement
q=0.9 %q=0.9 for shear prestressed structures
if Vu>0.5*q*(Vc + Vp)
    fprintf('transversal reinforcement is required')
    %transversal reinforcement
    Vs=(Vu/q)-Vc-Vp;
    %Minimum required transversal reinforcement(Av)
    %Calculate the required spacing for the transverse web reinforcement:
    %try Av=390mm^2
    Av=390;
    S=Av*fy*dv*cotd(tt)/Vs;
    %Checking the maximum spacing of shear reinforcement
    VU=abs(Vu-q*Vp)/(q*bv*dv);
    if VU<0.125*fc
        Smax=min(0.8*dv,600);
    elseif VU>0.125*fc
        Smax=min(0.4*dv,300);
    end
    if S>Smax
        S=Smax;
    elseif S<Smax
        S=S;
    end
    VS=Av*fy*dv*(cotd(tt))/S;
    if VS>Vs
```

```
        fprintf('minimum transverse reinforcement is satisfied')
    elseif VS<Vs
        fprintf('Av have to be increased')
    end
    %Nominal shear resistance
    Vn=min((Vc+Vs+Vp),0.25*fc*dv*dv+Vp);
    if Vn>Vu/q
        fprintf('web concrete will not crush prior to yielding of transverse
reinforcement')
    elseif Vn<Vu/q
        fprintf('web concrete will crush prior to yielding of transverse
reinforcement')
    end
elseif Vu<0.5*q*(Vc+Vp)
    fprintf('transversal reinforcement is not required')
end
 %%check longitudinal reinforcement requirement
 Vs=(Vu/q)-Vc-Vp;
 qf=1.0;
 qc=1.0;
 qv=0.9;
 if Aps*fps+As*fy>=(abs(Mu)/dv*qf)+(abs((Vu/qv)-Vp)-0.5*Vs)*cot(tt)
     fprintf('minimum longitudinal reinforcement requirement is satisfied')
 elseif Aps*fps+As*fy<=(abs(Mu)/dv*qf)+(abs((Vu/qv)-Vp)-0.5*Vs)*cot(tt)
     fprintf('minimum longitudinal reinforcement requirement is not satisfied')
 end
```

Step- 9 Calculate deflection and chamber

Algorithm A-0-8 Deflection and Camber

```
%assuming a parabolic tendon layout
Mc=Pi*e;
%As opposed to load deflection, camber is usually referred to as reversed
deflection and is caused by prestressing.
DD=((L^2)*(5*Mc))/(8*Ec*Ixx*6);
%deflection due to the applied load(gravity load)
D1=(5*PLL*L^4)/(384*Ec*Ixx);
if D1<L/360
    fprintf('deflection limit is satisfied')
elseif  DT>L/360
    fprintf('deflection limit is not satisfied, so you have to increase the
depth')
end
```

APPENDIX: B MULTI-OBJECTIVE OPTIMIZATION CODE

MATLAB Code for Post Tensioned Box-Girder Pedestrian Bridge

Algorithm B-0-1 Code for Design Objectives

```
function F=objval_(x)
%%%%x1,x2,x3,x4,x5,x6,x7 represents depth, thickness of flange, thickness of
web, bottom flange width, bottom flange thickness, length of overhang and area
of shear reinforcement.
x1 = x (:1); x2 = x(:,2); x3 = x(:,3); x4 = x(:,4); x5 = x(:,5); x6 = x(:,6);
x7 = x(:,7);
% Maximum factored moment Mu
Mu=(9727775195120271606445312 5.*x2)./2305843009213693952 +
(2161727821137838134765625.*x5)./2305843009213693952 -
(2161727821137838134765625.*x6)./2305843009213693952 +
(8646911284551352539062 5.*x3.*x4)./9223372036854775808 +
(8646911284551352539062 5.*x6.*(x1 - x2))./4611686018427387904 +
1032956869532765388936914062 5./576460752303423488;
%% factored moment resistance Mn
Mn=(191250.*x1 - ((2067152228963057716369628906 25.*x6)./576460752303423488 -
(2067152228963057716369628906 25.*x5)./576460752303423488 -
(9302185030333759723663330078125.*x2)./576460752303423488 -
(8268608915852230865478515625.*x3.*x4)./2305843009213693952 -
(8268608915852230865478515625.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
9877650064907069031709240722656 25./144115188075855872).^(1./2)).*((19.*x1)./20
- (97.*(191250.*x1 + (10.*((1945555039024054321289062 5.*x2)./576460752303423488
+ (4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(1729382256910270507812 5.*x3.*x4)./2305843009213693952 +
(1729382256910270507812 5.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
1505046700471877632273828125./144115188075855872)).*(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
```

```
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) -
((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2) -
1855125./14)).*(280.*((56.*((19455550390240543212890625.*x2).*(57646075230342348
8 + (43234556422756762695312 5.*x5).*(576460752303423488 -
(43234556422756762695312 5.*x6).*(576460752303423488 +
(17293822569102705078125.*x3.*x4).*(2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2)).*(1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872)).*(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
```

```
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + 4552972117138285./34359738368)))
- (((19.*x1)./20 - (97.*(191250.*x1 + (1649.*x6)./56 +
(10.*((19455550390240543212890625.*x2)./576460752303423488 +
(43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872))./(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
```

```
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) -
((20671522289630577163696289062 5.*x6)./576460752303423488 -
(20671522289630577163696289062 5.*x5)./576460752303423488 -
(93021850303337597236633300781 25.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407226 5625./144115188075855872).^(1./2) -
1855125./14)).*(280.*((56.*((19455550390240543212890625.*x2)./5764607523034234
88 + (43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872))./(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
```

```
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) +
4552972117138285./34359738368))).*((10416.*(191250.*x1 + (1649.*x6)./56 +
(10.*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872)).*(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
```

```
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) -
((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2) -
1855125./14))./(19.*x1.*((56.*((19455550390240543212890625.*x2)./57646075230342
3488 + (4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(1729382256910270507812 5.*x3.*x4)./2305843009213693952 +
(1729382256910270507812 5.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872)).*(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
```

```
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + 4552972117138285./34359738368)) -
1860).*((19455550390240543212890625.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*(1674.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
```

```
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500))+x2.*((1649.*x6)./56 -
1855125./14).*(x2./2 - (97.*(191250.*x1 + (1649.*x6)./56 +
(10.*((1945555039024054321289062 5.*x2)./576460752303423488 +
(4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872))./(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) -
((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
```

```
36576562500.*x1.^2 -
9877650064907069031709240722656 25./144115188075855872).^(1./2) -
1855125./14)).(280.*((56.*((19455550390240543212890625.*x2)./57646075230342348
8 + (4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872)).(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + 4552972117138285./34359738368)));
a=(97.*(191250.*x1 + (10.*((19455550390240543212890625.*x2)./576460752303423488
+ (4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
```

```
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + (85.*x2.*(2.*x6 - 4500))./2 -
((206715222896305771636962890625.*x6)./576460752303423488 -
(206715222896305771636962890625.*x5)./576460752303423488 -
(930218503033375972366333007 8125.*x2)./576460752303423488 -
(826860891585223086547851 5625.*x3.*x4)./2305843009213693952 -
(826860891585223086547851 5625.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2))).*(140.*((1649.
*x6)./28 + (56.*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
```

```
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500))));
de=-((191250.*x1 - ((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2)).*(x1 - 35) -
(19.*x1.*((10416.*(191250.*x1 +
(10.*((19455550390240543212890625.*x2)./576460752303423488 +
(43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
```

```
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872))./(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + (85.*x2.*(2.*x6 - 4500))./2 -
((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2))).*(19.*x1.*((16
49.*x6)./28 + (56.*((19455550390240543212890625.*x2)./576460752303423488 +
(43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
```

```
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)))) -
1860).*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
```

```
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872))./(33480.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) +
500)))./(((20671522289630577163696289 0625.*x6)./576460752303423488 -
(20671522289630577163696289 0625.*x5)./576460752303423488 -
(93021850303337597236633300 78125.*x2)./576460752303423488 -
(82686089158522308654785156 25.*x3.*x4)./2305843009213693952 -
(82686089158522308654785156 25.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
98776500649070690317092407 2265625./144115188075855872).^(1./2) - 191250.*x1 +
(((10416.*(191250.*x1 +
(10.*((19455550390240543212890625.*x2)./576460752303423488 +
(43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
```

```
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872))./(9.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) + (85.*x2.*(2.*x6 - 4500))./2 -
((206715222896305771636962890625.*x6)./576460752303423488 -
(206715222896305771636962890625.*x5)./576460752303423488 -
(930218503033375972366333007 8125.*x2)./576460752303423488 -
(826860891585223086547851 5625.*x3.*x4)./2305843009213693952 -
(826860891585223086547851 5625.*x6.*(x1 - x2))./1152921504606846976 +
36576562500.*x1.^2 -
987765006490706903170924072265625./144115188075855872).^(1./2))).*(19.*x1.*((16
49.*x6)./28 + (56.*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
```

```
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*(171.*x1.*(((500000.*x5)./3
- (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)))) -
1860).*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
```

```
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872))./(1674.*(((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)));
dv=max(max((de-0.5*a),(0.9*de)),(0.72*x1));
%%Contribution of concrete to nominal shear resistance
fpu=1860;
Es=200000;
Ep=197000;
fpo=0.7*fpu;
Nu=0;
asn=314.1593;
As=(6375*x1)/14 - ((20671522289630577163696289 0625*x6)/576460752303423488 -
(20671522289630577163696289 0625*x5)/576460752303423488 -
(93021850303337597236633300 78125*x2)/576460752303423488 -
(82686089158522308654785156 25*x3*x4)/2305843009213693952 -
(82686089158522308654785156 25*x6*(x1 - x2))/1152921504606846976 +
36576562500*x1^2 -
98776500649070690317092407 2265625/144115188075855872)^(1/2)/420;
Aps=((19455550390240543212890625*x2)/576460752303423488 +
(43234556422756762695312 5*x5)/576460752303423488 -
(43234556422756762695312 5*x6)/576460752303423488 +
(17293822569102705078125*x3*x4)/2305843009213693952 +
(17293822569102705078125*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
```

```
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
150504670047187763227382812 5/144115188075855872)/(1674*(((500000*x5)/3 -
(500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2
+ 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 +
100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 +
375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) + 500));
Aps1=-((194555503902405432128906 25*x2)/576460752303423488 +
(432345564227567626953125*x5)/576460752303423488 -
(432345564227567626953125*x6)/576460752303423488 +
(172938225691027050781 25*x3*x4)/2305843009213693952 +
(172938225691027050781 25*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
```

```
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
150504670047187763227382812 5/144115188075855872)/(1674*((528905046081400263933952*(((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 -
100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) +
500)*((19755856932175142875*((19455550390240543212890625*x2)/576460752303423488
+ (432345564227567626953125*x5)/576460752303423488 -
(432345564227567626953125*x6)/576460752303423488 +
(17293822569102705078125*x3*x4)/2305843009213693952 +
(17293822569102705078125*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
150504670047187763227382812 5/144115188075855872))/(64420386271395 84*(((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2
+ 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 +
100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 +
375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
```

```
100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) + 500)) -
197000)*(((24319437987800679016113281 25*x2)/144115188075855872 +
(5404319552844595336914062 5*x5)/144115188075855872 -
(5404319552844595336914062 5*x6)/144115188075855872 +
(2161727821137838134765625*x3*x4)/576460752303423488 +
(2161727821137838134765625*x6*(x1 - x2))/288230376151711744 +
1050394244339990528342285 15625/36028797018963968)/((500000*x5)/3 -
(500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2
+ 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 +
100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 +
375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912) +
((1945555039024054321289062 5*x2)/576460752303423488 +
(4323455642275676269531250 5*x5)/576460752303423488 -
(4323455642275676269531250 5*x6)/576460752303423488 +
(1729382256910270507812 5*x3*x4)/2305843009213693952 +
(1729382256910270507812 5*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
```

```
150504670047187763227382812 5/144115188075855872)/((((500000*x5)/3 -
(500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2
+ 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 +
100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 +
375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) +
500)*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) +
(250000*((19455550390240543212890625*x2)/576460752303423488 +
(43234556422756762695 3125*x5)/576460752303423488 -
(43234556422756762695 3125*x6)/576460752303423488 +
(17293822569102705078125*x3*x4)/2305843009213693952 +
(17293822569102705078125*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
150504670047187763227382812 5/144115188075855872))/((((500000*x5)/3 -
(500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2
+ 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 +
100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 +
375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
```

```
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) +
500)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5 -
100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))))/(935611865750292198310262627857*((19455550390240
543212890625*x2)/576460752303423488 +
(432345564227567626953125*x5)/576460752303423488 -
(432345564227567626953125*x6)/576460752303423488 +
(17293822569102705078125*x3*x4)/2305843009213693952 +
(17293822569102705078125*x6*(x1 - x2))/1152921504606846976 -
(5*2^(1/2)*((500000*x5)/3 - (500000*x6)/3 + 4500*x2*(x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 2*(50*x5 - 50*x6)*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3)
+ 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) +
100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 - x2))/6 + 10000*(x2 - (10000*x2
+ (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) +
x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 -
x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 + (10000*x2 + (100*x5 - 100*x6)*(x2
+ 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) +
2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 +
8947848533333333/536870912))/(2*(x1 - (10000*x2 + (100*x5 - 100*x6)*(x2 +
100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) +
1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))) +
150504670047187763227382812 5/144115188075855872)) -
10463609024031743/14724659719176192)*(((500000*x5)/3 - (500000*x6)/3 +
4500*x2*(x2/2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 +
2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5
- 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000))^2 + 2*(50*x5 - 50*x6)*(x2 -
(10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1
- x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 +
```

```
2*x6*(x1 - x2) + 10000) + 100/3)^2 + (x3*x4^3)/12 + 375*x2^3 + (x6^3*(x1 -
x2))/6 + 10000*(x2 - (10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 +
2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5
- 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000) + 100/3)^2 + x3*x4*(x4/2 - x1 +
(10000*x2 + (100*x5 - 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1
- x2) + x3*x4*(x1 - x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 +
2*x6*(x1 - x2) + 10000))^2 + 2*x6*(x1 - x2)*(x1/2 + x2/2 - (10000*x2 + (100*x5
- 100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))^2 + 8947848533333333/536870912)/((x1 - (10000*x2 + (100*x5 -
100*x6)*(x2 + 100/3) + 2250*x2^2 + 2*x6*(x1/2 + x2/2)*(x1 - x2) + x3*x4*(x1 -
x4/2) + 1000000/3)/(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) +
10000))*(4500*x2 + 100*x5 - 100*x6 + x3*x4 + 2*x6*(x1 - x2) + 10000)) + 500));
Vu=(11828974637266250273437 5.*x2)./36893488147419103232 +
(2628661030503611171875.*x5)./36893488147419103232 -
(2628661030503611171875.*x6)./36893488147419103232 +
(105146441220144446875.*x3.*x4)./147573952589676412928 +
(105146441220144446875.*x6.*(x1 - x2))./73786976294838206464 +
144552397527680999768328 75./9223372036854775808;
Vp=-(2501.^(1./2).*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872)).*(2501.*((528905046081400263
933952.*(((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2
+ (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
```

```
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)./((x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000))
+
500).*((19755856932175142875.*((19455550390240543212890625.*x2)./576460752303423
488 + (43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872))./(6442038627139584.*(((50000
0.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3
+ (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
```

```
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500)) -
197000).*(((24319437987800679016113281 25.*x2)./144115188075855872 +
(54043195528445953369140625.*x5)./144115188075855872 -
(54043195528445953369140625.*x6)./144115188075855872 +
(21617278211378381347656 25.*x3.*x4)./576460752303423488 +
(21617278211378381347656 25.*x6.*(x1 - x2))./288230376151711744 +
10503942443399905283422851 5625./36028797018963968)./((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912) +
((19455550390240543212890625.*x2)./576460752303423488 +
(43234556422756762695312 5.*x5)./576460752303423488 -
(43234556422756762695312 5.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
```

```
150504670047187763227382812 5./144115188075855872)./((((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) +
(250000.*((194555503902405432128906 25.*x2)./576460752303423488 +
(432345564227567626953125.*x5)./576460752303423488 -
(432345564227567626953125.*x6)./576460752303423488 +
(172938225691027050781 25.*x3.*x4)./2305843009213693952 +
(172938225691027050781 25.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)).*(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382812 5./144115188075855872)).*((((500000.*x5)./3 -
(500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
```

```
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 +
(x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500).*((500000.*x5)./3 - (500000.*x6)./3
+ 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) +
2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) +
1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) +
10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 + (x6.^3.*(x1 - x2))./6 +
10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 +
2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) +
1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)
+ 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912)))./(935611865750292198310262627857.*((19455550390
240543212890625.*x2)./576460752303423488 +
(4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(17293822569102705078125.*x3.*x4)./2305843009213693952 +
(17293822569102705078125.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
```

```
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) + 150504670047187763227382 8125./144115188075855872)) -
104636090240317 43./14724659719176192).*(((500000.*x5)./3 - (500000.*x6)./3 +
4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) +
2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) +
1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) +
10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3 + (x6.^3.*(x1 - x2))./6 +
10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 +
2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) +
1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)
+ 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 - (10000.*x2
+ (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500));
ex=(((Mu./dv)+0.5.*Nu+abs(Vu-Vp)-Aps.*fpo))/(Es.*As+Ep.*Aps1);
B=4.8/(1+750*ex);
es=ex;
q=0.9;
tt=29+3500.*es;
bv=2.*x6;
fc=50;
Vc=0.083.*B.*(fc).^0.5.*bv.*dv;
Vs=(Vu./q)-Vc-Vp;
fy=420;
L=50000; W=4500;
S=x7.*fy.*dv.*cotd(tt)./Vs;
VU=abs(Vu-q*Vp)/(q*bv*dv);
    if VU<0.125*fc
        Smax=min(0.8*dv,600);
    elseif VU>0.125*fc
        Smax=min(0.4*dv,300);
    end
    if S>Smax
        S=Smax;
    elseif S<Smax
        S=S;
    end
    VS=x7*fy*dv*(cotd(tt))/S;
    n=As/asn;
Vn=min((Vc+VS+Vp),0.25*fc*bv*dv+Vp);
V=225000000.*x2 + 5000000.*x5 - 5000000.*x6 + 50000.*x3.*x4 + 100000.*x6.*(x1 -
x2) + 500000000;
```

```
N11=(2199023255552.*((194555503902405432128906 25.*x2)./576460752303423488 +
(4323455642275676269531 25.*x5)./576460752303423488 -
(4323455642275676269531 25.*x6)./576460752303423488 +
(1729382256910270507812 5.*x3.*x4)./2305843009213693952 +
(1729382256910270507812 5.*x6.*(x1 - x2))./1152921504606846976 -
(5.*2.^(1./2).*((500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
375.*x2.^3 + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 +
8947848533333333./536870912))./(2.*(x1 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2
+ 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000))) +
150504670047187763227382 8125./144115188075855872))./(671498568618501303.*(((500
000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 + 375.*x2.^3
+ (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 - 100.*x6).*(x2 +
100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 -
x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2)
+ 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912)./((x1 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).*(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)) + 500));
f1=(V).*5000*10.^-9 + ((41.506.*N11*L*1102)*10^-6)+(n*W*2.42*70.04*10.^-
3)+(((L./S.*2*(W+x2).*0.62)*10.^-3+2.*((x1-x2)./S.*(L.*0.62))).*74.22*10^-3);
f2=(0.9*Vn/Vu);
f3=(Mn/Mu);
f4=min((0.9*Vn/Vu),(Mn/Mu));
F=[f1,-f4];
```

```
end
```

Algorithm B-0-2 Code for Design Constraint

```
function[Cineq,Ceq]=nonlcon(x)
L=50000; W=4500;

x1 = x(:,1); x2 = x(:,2); x3 = x(:,3); x4 = x(:,4); x5 = x(:,5); x6 = x(:,6);
x7 = x(:,7);

At = 4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000;

Ytot =(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 +
2.*x6.*(x1./2 + x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) +
1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) +
10000);

Ixx =(500000.*x5)./3 - (500000.*x6)./3 + 4500.*x2.*(x2./2 - (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*(50.*x5 - 50.*x6).*(x2 - (10000.*x2
+ (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + (x3.*x4.^3)./12 +
(375.*x2.^3) + (x6.^3.*(x1 - x2))./6 + 10000.*(x2 - (10000.*x2 + (100.*x5 -
100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1 - x2) +
x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 + x3.*x4 +
2.*x6.*(x1 - x2) + 10000) + 100./3).^2 + x3.*x4.*(x4./2 - x1 + (10000.*x2 +
(100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 + x2./2).*(x1
- x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 - 100.*x6 +
x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 2.*x6.*(x1 - x2).*(x1./2 + x2./2 -
(10000.*x2 + (100.*x5 - 100.*x6).*(x2 + 100./3) + 2250.*x2.^2 + 2.*x6.*(x1./2 +
x2./2).*(x1 - x2) + x3.*x4.*(x1 - x4./2) + 1000000./3)./(4500.*x2 + 100.*x5 -
100.*x6 + x3.*x4 + 2.*x6.*(x1 - x2) + 10000)).^2 + 8947848533333333./536870912;

Material selection
%AASHTO 2.4 concrete strength should be a minimum of 40Mpa
fc=50;
%Tensile strength , for most concretes the direct tensile strength
fr=0.62*fc^0.5 ,
%for prestressing strands the tensile straingth is fpu= 1860MPA
fr=0.62*(fc^0.5);
fpu=1860;
fy=420;
%yield strength for strand is given by 0.9*fpu for low relaxiation strand
fpy=0.9*fpu;
%modules of elasticity for steel reinforcement shall be assumed
%Es=200,000MPa , for presstressong strand Ep=197,000MPa
Es=200000;
Ep=197000;
%modules of elasticity of concrete
```

```
Ec=4800*(fc^0.5);
%unit weightof a concrete(rc) ,refer AASHTO table 3.5.1.1 in kN/m3
rc=24*10^-6;
%%%%%%% step 2 Material selection
%AASHTO 2.4 concrete strength should be a minimum of 40Mpa
fc=50;
%Tensile strength , for most concretes the direct tensile strength
fr=0.62*fc^0.5 ,
%for prestressing strands the tensile straingth is fpu= 1860MPA
fr=0.62*(fc^0.5);
fpu=1860;
fy=420;
%yield strength for strand is given by 0.9*fpu for low relaxiation strand
fpy=0.9*fpu;
%modules of elasticity for steel reinforcement shall be assumed
%Es=200,000MPa , for presstressong strand Ep=197,000MPa
Es=200000;
Ep=197000;
%modules of elasticity of concrete
Ec=4800*(fc^0.5);
%unit weightof a concrete(rc) ,refer AASHTO table 3.5.1.1 in kN/m3
rc=24*10^-6;

Dead load due to the self waight of concrete (PDL1)
PDL1=(At)*rc;
%barrier rail weight acting at service stage after all loos is 11.5N/mm
Pb=11.5;
%thickness of the wearing surface of 75mm
twer=75;
% distributed load of the wearing serface having density of 2250kg/m3
rw=22.5*10^-6;
Wb=200;
DW=(W-2*Wb)*twer*rw;
%total dead load due to unfactured self weaight ,barier and wearing weight
PDL=PDL1+DW+Pb;
%determination of the live load LL and dynamic load allowance IM
%AASHTO-3.6.2.1 state that dynamic load allowance shall not be applied to
%pedistrian loads or IM=1.0 should be used
%and for bridge only for pedistrian and biycle traffic shall be designed
%for a live load of 4.1*10^3 pa
PLL=18.45;
% for pedistrian load we will not concider IM
%load combianation
% service I check compressive stress in prestressed concrete component for
Q1=1.00*(PDL)+1.00*(PLL);
%service IIIcheck tensile stresss in prestressed concrete component
Q2=1.00*(PDL)+0.80*(PLL);
%check resistance for strength I
Q3=1.25.*(PDL)+1.5*DW+1.75.*(PLL);
%%%%%%% Determining the prestressing force
% the efficiency of the cross section p
P=Ixx./(At.*Ytot.*(x1-Ytot));
%permisible concrete stress (fa) and permisible tensile stress (fta)
fa=0.6*fc;
fta=0.5*fc^0.5;
```

```
%Moment requered to produce the permissible compresive stress at bottom most
fiber of the girder
c1=Ytot;
c2=x1-Ytot;
Mbc=(fa*Ixx)./c2;
%Moment requered to produce the permissible tensile stress at bottom most fiber
of the girder
Mbt=-(fta*Ixx)./c2;
%Moment requered to produce the permissible compresive stress at top most fiber
of the girder
Mtc=(fa*Ixx)./c1;
%Moment requered to produce the permissible tension stress at top most fiber of
the girder
Mtt=-(fta*Ixx)./c1;
%The mid span bending moment due to the applied load at service limit stateI
M=(Q2*L^2)/8;
%moment due to the own weight at mid span
MD=((PDL1+Pb)*L^2)/8;
%moment due to the wearing serface at mid span
Mw=((DW)*L^2)/8;
%moment due to the live load  at mid span
ML=((PLL)*L^2)/8;
% the center of gravity o the strand at mid span is assumed to be located
% at 5% of the girder heignt
ybs=x1*0.05;
e=500;

Prestressing Force
%minimum prestressing force requered to limite the bottem gieder tension
Fbmin=(M+Mbt)./(P.*c1+e);
%Max prestressing force requered to limite the bottem gieder tension
Fbmax=(M+Mbc)./(P.*c1+e);
%Max prestressing force requered to not exceed the minimum tension in top
girder tension
Ftmax=(M-Mtt)./(e-P.*c2);
%Min prestressing force requered to not exceed the minimum compression in top
gieder tension
Ftmin=(M-Mtc)./(e-P.*c2);
%the preliminary prestressing force is usually determined on the basis of the
service limit state III load condition at mid span
F=(M+Mbt)./(P.*c1+e);
%limits of eccentricity with regards to top of girde
emaxt=((M-Mtt)./F)+P.*c2;
emint=((M-Mtc)./F)+P.*c2;
%limits of eccentricity with regards to bottom of girde
emaxb=((M+Mbc)./F)-P.*c1;
eminb=((M+Mbt)./F)-P.*c1;
%Area of the strand (Ast)having tensile stangth of 1860 Mpa and 15.24 diameter
and
%Area requered dor the given value of prestressing force (Aps),number of
strands requered (N)
Ast=((15.24).^2.*pi)./4;
Aps=F./fpy;
N11=Aps./(Ast);
```

```
calculate prestress loss
%1 friction loss fpf , AASHTO art.5.9.5.2.2b
%ep is the distance b/n two centroid points Lp is the horizontal distance
%b/n two control points , K is the web friction coefficient 0.0002 ,
%U is the coefficient of friction 0.25
fpj=0.9.*fpu;
K=0.0002;
U=0.25;
ep=e;
Lp=0.5.*L;
a=2.*ep./Lp;
x=780;
fpf=fpj.*(1-(exp(-(K.*x+U.*a))));
%2 Anchorage set loss fpa
%for an anchor set thickness of L1=10mm and E=200,000mpa ,Lpf=50m
Lpf=50000;
L1=10;
Lpa=(Ec.*L1.*Lpf./fpf).^0.5;
%3 elastic shortning loss fpes AASHTO 5.9.5.3.b
%fcgp=(F./A)+(F.*e.^2./Ixx)+(M.*e./Ixx) , where M is the moment due to own
weight,
%fcgp is the sum of the concrete stress at the center of graity of the
%prestressing force after jacking for post tensioned member, for simply
%supported structure calculated at the center section of the span
fcgp=(F./At)+(F.*e.^2./Ixx)+((MD+Mw).*e./Ixx);
fpes=(N11-1).*Ep.*fcgp./(2.*N11.*Ec);
if Lpa>Lpf
fpa=0; %no anchorage set loss
elseif Lpa<Lpf
f=2.*fpf.*Lpa./Lpf;
%assuming horizontal dostance X=Lpa
fpa=f;
end
%4 time dependant loss
%AASHTO provides s tsble to estimate the accumulated effect of time dependant
%lossed resulting from the creep and shrinkage of concrete and relaxation of
%the steel tedons .there for fptm=145MPa
fptm=145;
fpt=fpf+fpa+fpes+fptm;
% To determone the jacking force Pj, after loss the initial prestress force
% coefficient Fpci and final prestress coefficient Fpcf
Fpci=1-(fpf+fpa+fpes)./fpj;
Fpcf=1-(fpt./fpj);
%the requered prestressing force at transfer (befor any loss)
Pi=F./Fpcf;
Aps1=Pi./fpy;
N2=Aps1./(Ast);
%%check prestressing stress limit at service limit state
%the initial stress in prestressing steel before transfer <=0.75*fpu
fpbt=0.75.*fpu;
fpe=fpbt-fpt;
%check concret stress for service limit state
%checking concrete stress for service limite state
%Stress at the top of CIP box girder (ftop) due to the own weight should be <fa
=0.6*fc Mpa
```

```
ftop=(MD+Mw).*c1./Ixx;

check concrte strength at sevice limit state I at bottom fiber after
loss
M1=Q1.*L.^2./8;

fcsb=Fpcf.*Pi./At +Fpcf.*Pi.*e.*c2./Ixx -M1.*c2./Ixx;

% Check tensile stresses at bottom of the girder under service III
% fbot<fta
M2=Q2.*L.^2./8;
fbot=(Pi.*Fpcf./At)+(Fpcf.*Pi.*e.*c2./Ixx)-(M2.*c2./Ixx);
%%%%% step 6 Design for strength limite state-flexure
%Maximum factored moment Mu
Mu=1.25.*MD+1.5.*Mw+1.75.*ML;
%Average prestressing steel stress
fpy=0.9.*fpu;
K=2.*(1.04-(fpy./fpu));
B1=0.85-((0.05)*((fc-28)/7));
if B1<0.65, B1= 0.65;
disp('if')
elseif B1>0.65 ,B1=B1;
disp('elseif')
end
%%assume that the compressive area is rectangular and fs=fy the distance from
neutral axis to extreme compressive fiber is as follows
fs=fy;
dp=x1-0.05*x1;
cc=25;
d=20;
ds=x1-cc-d/2;
As=(0.85.*fc.*W.*x1-((0.85.*fc.*W.*x1).^2 -1.7.*fc.*W.*Mu).^0.5)./fy;
Asmin=0.002.*At;
if As < Asmin, As= Asmin;
disp('if')
elseif As >Asmin ,As=As;
disp('elseif')
end
C=(Aps.*fpu+As.*fs-0.85.*x2.*fc.*(W-
2.*x6))./(0.85.*fc.*B1.*(2.*x6)+K.*Aps.*(fpu./dp));
a=B1.*C;

The average stress in the prestressing steel fps
fps=fpu.*(1-(K.*C./dp));
de=(Aps.*fps.*dp+As.*fy.*ds)./(Aps.*fps+As.*fy);
%%factored moment resistance
Mn=(Aps.*fps.*(dp-0.5.*a)+As.*fy.*(dp-0.5.*a)+0.85.*fc.*(W-2.*x6).*x2.*(0.5.*a-
0.5.*x2));
%%check the assumption that the section is tension controlled.
%%et>0.005
dt=x1-0.05.*x1;
et=0.0030.*(dt-C)./C;
%%%% minimum reinforcement
%%the amounte of prestressed tensile reinforcement at any section requeres that
%%flexural resistance Mr, equals the leasser of i,1.33*Mu and Mcr
```

```
fr=0.62.*fc.^0.5;
fpe=(Pi./At)+(F.*e.*c2./Ixx);
fd=(MD+Mw).*c2./Ixx;
Mcr=Ixx.*(fr+fpe-fd)./Ytot;
%%%% step-7 design for shear-strength limit state I
Vu=0.95.*Q3.*L.*0.5;
%critical section for shear design
%the effective depth for shear
de=((Aps.*fps.*dp)+(As.*fy.*ds))./((Aps.*fps)+(As.*fy));
dva=[(de-0.5.*a) (0.9.*de) (0.72.*x1)];
dv=min(dva);
O=atan((e./(0.5.*L)));
Vp=Pi.*sind(O.*180./pi);
if Mu> abs((Vu-Vp).*dv)
    fprintf('ok')
end
%%Contribution of concrete to nominal shear resistance
fpo=0.7.*fpu;
Nu=0;
ex=(((Mu./dv)+0.5.*Nu+abs(Vu-Vp)-Aps.*fpo))./(Es.*As+Ep.*Aps1);
es=ex;
B=4.8./(1+750.*ex);
tt=29+3500.*es; %%AASHTO 5.8.3.4.2.3
%%concrete contribution for shear
bv=2.*x6;
Vc=0.083.*B.*(fc).^0.5.*bv.*dv;

Requerment for shear reinforcement
q=0.9
%transversal renforcement
 Vs=(Vu./q)-Vc-Vp;
 %Minimum requerd transversal reinforcement(Av)
 %Calculate the required spacing for the transverse web reinforcement:
 %try Av=390mm^
 Av=x7;
 S=Av.*fy.*dv.*cotd(tt)./Vs;
 %Checking the maximum spacing of shear reinforcement
 VU=abs(Vu-q*Vp)./(q*bv*dv);
 if VU<0.125*fc
   Smax=min(0.8.*dv,600);
 elseif VU>0.125*fc
   Smax=min(0.4.*dv,300);
 end
 VS=Av.*fy.*dv.*(cotd(tt))./S;
 %Nominal shear resistance
 Vn=min((Vc+Vs+Vp),0.25.*fc.*bv.*dv+Vp);
 %%check longitudinal reinforcement requirement
 Vs=(Vu./q)-Vc-Vp;

Calculate deflection and chamber
%assuming a parabolic tendon layout
Mc=Pi.*e;
%As opposed to load deflection, camber is usually referred to as reversed
deflection and is caused by prestressing.
DD=((L.^2).*(5.*Mc))./(8.*Ec.*Ixx.*6);
```

```
%deflection due to the applied load(gravity load)
D1=(5.*Q1.*L.^4)./(384.*Ec.*Ixx);
DT=D1-DD;
DL=(5.*ML.*L.^2)./(8.*Ec.*Ixx.*6);
if DL<L/360
    fprintf('diflection limite is satisfied')
elseif  DL>L/360
    fprintf('diflection limite is not satisfied , so you have to increase the
depth')
end

cin1 = abs(sqrt(M./(0.27.*W.*fc)))-x(:,1);
cin2 = ftop-fa;
cin3 = x2-C;
cin4 = fcsb-0.45.*fc;
cin5 = fbot-0.5.*((fc).^0.5);
cin6 = fbot-fta;
cin7 = Asmin-As;
%{
cin8 = zeros(size(M)); cin8(1) = 0.65-B1;
%}
cin8 = zeros(size(M,1),0);   %should be present, but always positive
%cin9 = Lpf-Lpa;
cin9 = zeros(size(M,1),0);   %removed
cin10 = 0.5.*fpu-fps;
cin11 = C-0.42.*de;
cin12 = 0.005-et;
cin13 = 1-(Mn./Mu);
cin14 = (Mn./Mu)-3;
cin15 = (Mn./Mu)-3;
cin16 = (0.9*Vn./Vu)-3;
cin17 = fpe-0.8.*fpy;
%{
cin18 = abs((Vu-Vp).*dv)-Mu;
cin19 = Vs-VS;
cin20 = 0.5.*q*(Vc+Vp)-Vu;
cin21 = Vn-Vu./q;
cin22 = DL-L/360
cin23 = (abs(Mu)./dv.*qf)+(abs((Vu./qv)-Vp)-0.5.*Vs).*cot(tt)-
(Aps.*fps+As.*fy);
%}
cin18 = zeros(size(M,1),0);   %removed
cin19 = zeros(size(M,1),0);   %removed
cin20 = zeros(size(M,1),0);   %removed
cin21 = zeros(size(M,1),0);   %removed
cin22 = zeros(size(M,1),0);   %removed
cin23 = zeros(size(M,1),0);   %removed
Cineq=[cin1, cin2, cin3, cin4, cin5, cin6, cin7, cin8, cin9, cin10, cin11,
cin12, cin13, cin14,cin15, cin16, cin17, cin18, cin19, cin20, cin21, cin22,
cin23];
Ceq=[];
end
```

Algorithm B-0-3 Main Code for Pareto Set

```
%design of post-tensioned box girder bridge, its length is 50m
%%%%%%%step -1 determining cross section geometry
%structural depth D, the span of the bridge(L), width of the girder(W),the over
hanging width(Lc),botem flange width(bbot)
%Tickness of the web(tf) , thickness of the top flang(tf), thickness of the
bottom flange (tbf), the fillet height(s) ,moment of inertia of the section
(Ixx),
%Area of the sections(Ai),total area of the box structure (At) ,centroid of the
the areas (Yi), centroidal distance from the top fiber (Yii)
%Centroid of the box stracture (Yt),volume of the concrete (V),the effective
flange width(S)
%Unite waight of the normal concrete in kg/m^3(rc),gravity(g) ,width of the
barier(Wb)
%h2=height of the gieder form the bottom of the slan to the bottom of the
girder
%AASHTO 2.4 concrete strength should be a minimum of 50Mpa
W=4500;
Aeq=[];
Beq=[];
Aineq=[];
bineq=[];
lb=[1000,175,4500-2*1600,200,530,200,800];
ub=[3000,400,4500-2*530,400,1800,400,2000];
%function for calculation of objective funtwection and nonlinear constraints
fun = @objval_;
nlcon = @nonlcon;
npts=900;
opts_ga =
optimoptions('gamultiobj','Display','iter','PlotFcn',@gaplotpareto,'PopulationS
ize',2*npts);
N = rand(1e6, 7);
rval = lb + (ub-lb).*N;
%calculate constraints at each point, all at once#
val = nlcon(rval);
%see if it is even hypothetically possible to satisfy the constraints
minval = min(val);
disp(minval);
if any(minval > 0)
    fprintf('unsolved constraints\n');
else

[x_gal,fval_gal,~,gaoutput1]=gamultiobj(fun,7,Aineq,bineq,Aeq,Beq,lb,ub,@nonlco
n,opts_ga);
  fval_gal = fval_gal .* [1 -1];   %adjust back for negative because of
maximization
  disp("Total Function Count:"+gaoutput1.funccount);
  disp("Total Function Count:"+gaoutput1.funccount);
```

```
  fga = sortrows(fval_gal);
  fig1 = figure;
  ax1 = axes(fig1);




  hold(ax1, 'off')
  plot(ax1, fga(:,1),fga(:,2),'b--')
  xlabel(ax1, 'Cost in Birr')
  ylabel(ax1, 'Safety Factor')
  legend(ax1, 'Gamultiobj')
  hold(ax1, 'on')
end
```