BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF MECHANICAL ENGINEERING
FAKULTA STROJNÍHO INŽENÝRSTVÍ

INSTITUTE OF AEROSPACE ENGINEERING
LETECKÝ ÚSTAV

MULTI-OBJECTIVE OPTIMIZATION OF COMPLEX COMPOSITE STRUCTURES WITH VARIABLE STIFFNESS
VÍCECÍLOVÁ OPTIMALIZACE SLOŽITÝCH KOMPOZITNÍCH KONSTRUKCÍ S VARIABILNÍ TUHOSTÍ

DOCTORAL THESIS
DIZERTAČNÍ PRÁCE

AUTHOR — **Volodymyr Symonov, M.Sc.**
AUTOR PRÁCE

SUPERVISOR — **doc. Ing. Jaroslav Juračka, Ph.D.**
ŠKOLITEL

BRNO 2021

ABSTRAKT

Disertační práce se věnuje vývoji metodologií pro vice-cílovou optimalizací složitých kompozitních konstrukcí s proměnnou tuhosti. Vice úrovňový hybridní optimalizační algoritmus je založený na bázi hybridní optimalizační metody s využitím interpolační plochy odezvy, genetického algoritmu a jednoparametrické optimalizace. Pro strukturální analýzy je využit MKP software MSC Nastran. Nový genetický algoritmus a paralelní jednoparametrický optimalizační algoritmus na základě metody Zlatého řezu jsou vyvinuty pro metodologii. MKP software a vyvinuté optimalizační algoritmy jsou integrované pomoci komerčního optimalizačního softwaru Noesis Optimus od Noesis Solutions. Vyvinutá metodologie je ověřena pomoci testovací optimalizační úlohy.

KLÍČOVÁ SLOVA

Vice-cílová optimalizace, víceúrovňová, více parametrická, kompozitní konstrukce, proměnná tuhost, genetický algoritmus.

ABSTRACT

The thesis is dedicated to development of a multi-objective optimization methodology for complex composite structures with variable stiffness. A multi-level hybrid optimization algorithm is developed based on a hybrid optimization method with interpolating response surface, a Genetic Algorithm and a one-dimensional optimization. Finite element analysis software MSC Nastran is used for structural analyses. A new Genetic Algorithm and a parallel one-dimensional optimization algorithm based on “Golden section” method are developed for the methodology. The finite element analysis software and the developed optimization algorithms are integrated with help of a commercial optimization software Noesis Optimus by Noesis Solutions. The developed methodology is verified on an example optimization problem. The results of the problem optimization are compared to those obtained using previously developed methodologies.

KEY WORDS

Multi-objective optimization, multi-level, multi-parametric, composite structure, variable stiffness, genetic algorithm.

DECLARATION

Hereby I declare that the doctoral thesis "Multi-objective optimization of complex composite structures with variable stiffness" is the result of my original investigation and achievement made under the guidance of my supervisor and using professional literature and sources provided in references, which are listed at the end of this thesis.

AKNOWLEDGEMENTS

First, I would like to thank my teachers from primary school for their aspiration to nest into our minds not only knowledge but also strive and desire to perceive the world, develop ourselves not only intellectually but also spiritually, to be worthy people, be able to stem difficulties and move beyond the limits. A great thanks for your patience, eagerness and personal touch gifted to us. I would like to express my special gratitude to Irina Nikolaevna Kovaleva my teacher of mathematics, who dropped to me and my classmates a living spark of perception.

Of course, I would like to thank my high school teachers from National Aerospace University "Kharkov Aviation Institute"(KhAI) not only for their professionalism, knowledge and the door to the aerospace world opened to us but also for fostering, parting words and admonition, which helped us to become not only professional engineers but also true, strong and purposeful characters. Especially I would like to express my gratitude to my scientific advisor at National Aerospace University "Kharkov Aviation Institute"(KhAI) professor Iakiv Semenovich Karpov, D.Sc, who showed me the way to fascinating world of science and became for me a role model of a True Scientist and Real Man, which I still have been following.

For sure, I appreciate my scientific advisor, director of Institute of Aerospace Institute at Brno University of Technology, associate professor Jaroslav Juračka, Ph.D., for giving me an opportunity to study doctoral programme at BUT and for invaluable support and help with performing this thesis.

Also, I am thankful to my father and grandfather. By virtue of their vocation I was grown up in a friendly and well-knitted aeronautical community. I was lulled with roar of jet engines and smell of jet fuel, delighted with singing of skylarks, melt into the high peaceful dazzling blue sky over an airfield.

Of course, I would like to thank my close relatives, mother, brother and my wife Yana for their support and understanding.

CONTENTS

INTRODUCTION

Everything is possible. The impossible just takes longer.
— Dan Brown, ***Digital Fortress***

Every leaving creature or process in nature is the result of millions of years' long extremely complex multidisciplinary, multi-objective and multi-parameter optimization. Such an approach gives the leaving creatures and processes the abilities to be extremely tightly incorporated into their natural life environment. Since the human is the part of nature the creatures made by him also always pass the multistep optimization. New scientific discoveries, technologies and materials make the optimization process to do the next step on the way to the ideal artificial structures and systems.

Thus, the composite materials as the relatively new ones give the humankind the outstanding capabilities in improvement of structures in many technical fields.

Among many other advantages the laminates are extremely flexible in strength and stiffness tailoring, therefore, they are very good materials in sense of structural optimization.

The aerospace industry is the frontier of implementation of progressive advanced composite structures and their manufacturing technologies. Moreover, weight of aerospace structures is one of the most critical parameters for aerospace products, which directly influences their performances such as range, payload capacity, service coast, etc. Application of composite materials for space structures is already normal practice for a long time, which does not require to be introduced and explained. Fully composite airframes for small one- to four-seater airplanes and sailplanes are also not new in the market for almost four decades. It can be seen very well from the AERO Fiedrichshafen Global Show for General Aviation, where within the recent several years the most part of participants presented composite airplanes. The new trend emerged within the last decade is entering the world market by wide-body airliners, which have a very high rate of composite materials in their airframe structures. For example, the use of composites in Boeing 787 Dreamliner is 50% by weight, the Airbus A350 XWB airframe structure is 53% made of composites and the new Russian airliner MS-21 has whole composite wing. Therefore, minimization of weight of composite aerospace structures' is very important and timely problem nowadays.

The modern approach to the design, stress analysis and optimization of aerospace structures is hardly thinkable without applying CAD and FEA software. Nowadays, an ordinary aerospace engineer is experienced in several types of such software and has skills for proceeding through design process using it. Of course, almost each software has its own build-in optimization abilities, however very often they are limited especially in case of composites. Considering this situation, it is very timely to think of a flexible approach, which will allow to widen optimization possibilities for aerospace engineers, who design composite structures. Within these optimization approach CAD and FEA software can be used for design, static and dynamic analyses of any type of thin-walled structure. It is very important to make possible incorporating any type of CAD and/or FEA commercial or domestic software, which is traditionally used by a company, into the optimization procedure.

1 The state-of-the-art review

Optimization of a complex aerospace structure is a complex mathematical and engineering problem, which comprises many parameters (for large structures the number of parameters can be several thousands). Evidently, such problems cannot be solved effectively using simple and well-known direct methods, which include all the variables describing a complex structure. Nowadays, such problems are decomposed into several levels and/or sub problems. This approach is called multi-level optimization.

Multi-level optimization. Multilevel approaches are capable of breaking down the optimization problem into several optimization problems that can be solved separately in an iterative process. A hierarchical decomposition divides the problem into a system level problem and a set of uncoupled component level problems. There are also non-hierarchical decompositions that divide the problem into several parallel problems. For example, the most common form of this decomposition applied to composite materials consists of decoupling the optimization of the thicknesses from that of the fiber orientations. At one level, only the thickness is optimized, leaving the search for the best fiber orientation for each ply to the second level.

Hierarchical decomposition of design problems has been widely used for the design of complex systems made of traditional metallic materials [1], [2]. Schmit and Mehrinfar [3] were the first who extended and applied this method to the design of a hat-stiffened composite panel. It supposes dividing the design variables into system level and component level variables. At the system level, the weight of the structure is minimized subjected to system level constraints. At the component level, the detailed design variables are obtained by minimizing the change in the stiffness of the component subject to a set of local constraints. In order to weaken the coupling between the two levels and to ensure convergence of the overall design, an appropriate decomposition guided by insight into the physics of the problem and a proper modeling approach are necessary. For simple structural models, the integration of the two levels can be handled well; however, for complex configurations, finding an effective decomposition can be cumbersome and the resulting solution may be suboptimal.

Hierarchical decomposition may also be used to break down the system to several component-level optimization problems linked together through a global-level analysis [5]. In this case, no optimization is performed at the system-level, but the system-level analysis is used only to update the information at the component level.

Non-hierarchical decomposition [6] – [8] assume that at the upper level the optimization is applied to the entire structure, where the laminates thicknesses are the only design variables. At the lower level, the fiber orientation at each element is considered as design variables and the problem is to minimize the weight while constraining the change in stiffness of the element to a minimum. This ensured that the stiffness, and hence the load paths, within the overall structure do not change substantially, thus the continuity is preserved when the solution returned to the upper-level optimization.

There are two advantages in using decomposition techniques. First, by employing a decomposition technique and a multilevel algorithm the number of variables is reduced during the optimization process, which can significantly reduce the computational effort required. Second, by performing the optimization at two separate levels, one can choose an appropriate optimization method that is particularly efficient for each subproblem and benefits from advantages of two or more optimization methods. The implementation of decomposition technique also suffers from two main shortcomings. First, decomposition approaches require physical insight into the problem and needs establishing a close integration of the low levels, which has prevented their ready application with the black-box codes often used in the aircraft industry. Second, the efficiency of the method (i.e., convergence) and the quality of the solution (i.e., optimum or near-optimum) strongly relies on the decomposition; for instance, it is recognized that a minimum weight structure is not necessarily made up of a collection of minimum weight substructures. Furthermore, difficulties may arouse in convergence when the lower-level problems are described with highly non-linear functions in terms of design variables.

The hierarchical and non-hierarchical decomposition methods are successfully used by modern aircraft design companies, e.g., AIRBUS [9].

Hybrid methods. A hybrid method refers to an optimization algorithm that uses more than one optimization technique without decomposing the original problem into sub-problems. A hybrid method usually iteratively switches between two or more optimization methods in order to benefit from the advantages of each constituent method. Although this type of optimization technique is found promising for constant stiffness design of composite laminates [10], their application in variables stiffness design is dominated by multi-level methods, which besides having the potential of gaining benefits from more than one optimization technique, can also reduce the size of the problem.

Regardless of the optimization technique, efficiency (i.e., convergence rate and accuracy) of the optimization algorithm is generally reduced as the number of design variables increases. On the other hand, independent design of these arrangements may result in an impractical structure with structural discontinuities. Hence, part of the effort in formulating a variable stiffness design problem is dedicated to minimizing the number of design variables in the problem formulation and to maintaining the continuity in the structure. For this purpose, three approaches are known based on the use of a “patch design”, a set of “blending rules”, and a curvilinear definition of the variable properties.

Patch design. A ‘‘patch” defines a region within the structure where the lamination sequence is uniform. The use of a patch allows reduction of the number of design variables and the spatial variation of the lamination sequences; consequently, it eases consideration of manufacturing limitations and maintains the compatibility of the lamination sequences in adjacent regions. Both overlapping patches [11], [12] and non-overlapping patches [13] have been used to formulate a variable stiffness design problem. Usually in this formulation the patch geometry, which can strongly affect the efficiency of the final design, is defined a priori by the user.

Blending rules. Enforcing continuity constraints that limit the variation of the lamination sequences in adjacent elements is another method to achieve a blended structure. These constraints are generally known as blending rules. Examples of the blending rules are the ones used by Zabinsky et al. [14]. To design a tapered structure, the blending rules might be stated as follows: ‘‘starting from the key region, the number of plies in adjacent regions may be dropped if the required stiffness and strengths of these regions are satisfied.

Once a ply is dropped, it cannot be added back to the stacking sequence in later regions. Although able to provide a required level of continuity in the structure, this method increases the number of constraints and makes the optimization problem more complex. To reduce the complexity of the design problem, it is possible to combine the blending rules with the patch design approach which can significantly reduce the number of design variables.

Curvilinear parameterization. The use of a curvilinear function to describe the fiber path and the variation of the laminate thickness can also reduce the number of design variables, without compromising structural continuity. The curvilinear function is identified by a set of parameters in a pre-defined mathematical expression [15] or by interpolating a pre-defined function to prescribed key points [16]. For instance, Bezier splines might be used to represent laminates thicknesses and fiber angles. Using a curvilinear function to describe the fiber path significantly reduces the number of design variables and guarantees the structural continuity; however, the quality of the final solution strongly depends on the parameterization. Finding a parametric function that can accurately model a complex structural geometry is difficult.

Today many research teams are working on optimization of aircraft composite structures, [4], [17] – [18], [21], [23]. The most interesting articles are discussed below in detail.

P. Gasbarri et al. [4] proposed a hybrid multilevel approach for aeroelastic optimization of a composite wing-box, which is aimed to maximize the flatter speed of the wing structure. A hybrid two level procedure is used for a Multi-Disciplinary Optimization. The upper-level global aeroelastic optimization problem is solved by a deterministic approach based on Sequential Quadratic Programming and on the method of feasible directions. The upper-level optimization uses very few global quantities, such as the degree of anisotropy of the material called lamination parameters.

The lower level consists of independent subproblems at different stations along the wing. They are solved by a stochastic approach based on a genetic algorithm (GA), which optimize the lay-up angles to reconcile the upper-level independent variables. This reconciliation is improved further by a set of upper-level coordination constraints. Strength constraints for the composite panels are also applied at this level.

The structural analysis at the upper level is performed using a modified FE method, which is based on assuming a span-wise finite element discretization of the wing elastic displacement. This approach provides global chord-wise quantities such as flexural displacements, torsional rotations, mean-line curvature, etc., which are meaningful from the aerodynamic standpoint. The FE model uses one-dimensional elements for discretization of the wing semi-span.

The stiffness of the structure at this level is defined by the matrices D corresponding to each wing panel. Each matrix corresponds to laminate bending stiffness submatrix according to classical laminate theory. It is proposed to express these matrices as following:

$$D = D^{00} + H^{01}\zeta_3 + H^{02}\zeta_1 + H^{10}\zeta_4 + H^{12}\zeta_2 , \tag{1.1}$$

where D^{00} – matrix, which depends on lamina material properties and thickness only; H^{ij} – matrices of material constants which depends only on the lamina material properties; terms ζ_k are the functions of the orientation of the plies of laminate, of their thickness and of their stacking sequence.

The terms ζ_k are called the integral lamination parameters. They are taken as independent variables at the upper level. In case of symmetric orthotropic laminates two terms ζ_1 and ζ_2 are required to define a D matrix, since the matrices H^{01} and H^{10} are zero ones.

Such an approach allows splitting the optimization process at the lower level for different panels. The laminate stacking sequence, number of plies and/or laminate thickness, structural dimensions vary within the stiffness or integral parameters ζ_k found during global optimization for each panel. In such a way, panels can be optimized in parallel at the lower level.

Here at this level the objective function is a measure of the difference between the upper level design variables $\bar{\zeta}_1$ and $\bar{\zeta}_2$ and the functions ζ_1 and ζ_2 determined by prescribing the lower design variables (orientation angles of plies):

$$F = \frac{(\zeta_1 - \bar{\zeta}_1)^2}{{\zeta_1}^2 + {\bar{\zeta}_1}^2} + \frac{(\zeta_2 - \bar{\zeta}_2)^2}{{\zeta_2}^2 + {\bar{\zeta}_2}^2}. \tag{1.2}$$

No analysis is performed at this level.

The main advantage of such an approach is in choosing the terms ζ_k as upper-level variables. This allows to describe the entire wing structure with a very few parameters. However, these variables are not independent from each other. This fact means not every pair of ζ_1 and ζ_2 is feasible. From the other hand these variables have not a clear physical meaning what is not convenient from the engineering point of view.

Another drawback of the approach is the upper-level analysis. Since it requires a special FE method, common FEA software cannot be used here. From the other hand the FE model used here is limited to long and narrow structures like wing or fuselage. The other structures cannot be analyzed by this FE method.

At last, there is no analysis at the lower level. That means whole structure should be analyzed at the upper level at each optimization step to check the constraints. However, with proposed simple FE model such an approach seems not to be computationally expensive, in case of more complex FE model it will be very expensive to analyze an FE model of a complex structure at each optimization step.

S. Guo [17] proposed an algorithm for achieving an optimal design for an aerobatic aircraft wing structure to meet the lightweight, strength and aeroelastic design requirements.

In this investigation, the wing box between the front and rear spar was assumed to be the primary structure of the wing. The components and surface after the rear spar were only counted in calculating the weight, inertia, and aerodynamic force of the wing. For stiffness and vibration analysis, the tapered wing box structure clamped at the root section was divided into several spanwise segments. Each of the wing box segments was modelled as a uniform thin-walled single-cell box beam and the whole wing box was modelled as an assembly of those beams.

The optimization was performed in two stages. In the first stage, effort was made to design and model a composite wing for a minimum weight structural option. An analytical method was used for vibration and aeroelastic analysis and the finite element (FE) method (MSC Patran/Nastran software) was employed for structural stress analysis.

In the aeroelastic optimization stage, flutter speed was chosen as the primary parameter in the objective function and the fiber orientation of the skin and spar webs laminates were taken as the design variables. Instead of the usual constrained optimization, the unconstrained aeroelastic tailoring was conducted separately from the stress analysis in this current investigation.

To solve the optimization problem, the Davidon–Fletcher–Powell variable metric method is employed as the optimizer whereas the Golden Section method based on polynomial interpolation is employed for the one-dimensional search.

An aerobatic aircraft was taken as a study example. The aircraft has a maximum take-off weight 1000 kg, dive speed 122 m/s and cruise speed 103 m/s. The design ultimate load factors are +7g and −5g.

The initial design of the wing was made of 8-ply (0.125 mm ply thickness) carbon/epoxy quasi-isotropic laminates for spars webs, ribs, stringers, and skins.

In each of the segments, the wing box was further divided into four laminate panels along its cross-section circumference representing the two skin covers and two spar webs.

Four optimization cases were proposed in the paper. Depending on the initial laminate layup, selection of plies and segments for tailoring, the number of design variables and optimization results are different for each case.

In the first case, each ply in all the six spanwise segments was set as an independent variable and different number of plies was selected each time in the optimization. Since the layup in each of the spanwise segments was tailored independently, the optimum layup in the segments along the span was different from each other. The number of design variables will vary from the minimum zero to the maximum 192 when eight plies were selected.

In the second case, each ply of the laminates was taken as an independent variable, but the layups of the six spanwise segments were kept uniform during the optimization. Although the spanwise layup tailoring was limited and the maximum number of design variables was reduced to 4×8, the flutter speed was steadily increased as more plies were being tailored.

The case 3 is similar to case 1, each ply of the laminates in the spanwise segments was selected as an independent design variable. The difference in this case is that the laminate layup of skins and spar webs is restricted to symmetric.

In the case 4, in addition to the symmetric laminate layup as specified in case 3, the layups in the spanwise segments were kept uniform. In this case, the number of design variables was reduced to the minimum (4×4). Nevertheless, greater flutter speed increase than that obtained in case 3 was achieved when different number of plies is tailored.

Finally, the author has compared the best optimized design to the original metallic wing box. The comparison has shown 40% weight saving can be achieved by using a thinner composite wing box of adequate strength. Without constraining the aeroelastic tailoring to a strength condition, significant increase of flutter speed up to 37% has been achieved.

In the proposed approach the thickness of the structural elements being tailored was constant and only the plies angles were varied. It restricts the optimization domain to the laminates of that constant thickness. Moreover, the approach does not use the benefits of thickness change along the wingspan and chord directions. These facts, for sure, lead to suboptimal results. Finally, the authors have not proposed any approach to reduce the number of variables.

Q. Zhao et al. [18] proposed optimization method for large-scale composite wing structures. The two-level optimization strategy is applied by the authors. Design requirements are adjusted at the system (upper) level according to structural deformation, while the layout is optimized at the subsystem (lower) level to satisfy the constraints from system level.

A dimensionless failure coefficient is employed to represent the ratio between critical load of a specific failure mode and the current load applied to the structure:

$$e_f = \sqrt{\left(e_s/e_{sp} - 1\right)^2 + \left(e_l/e_{lp} - 1\right)^2 + \left(e_g/e_{gp} - 1\right)^2}, \tag{1.3}$$

where e_s, e_l and e_g - actual failure coefficients of static strength, local buckling, and global buckling respectively; e_{sp}, e_{lp} and e_{gp} - corresponding coefficients that the structure is required to achieve.

The closer e_f to zero, the higher structural efficiency could be achieved. Therefore, the equation (1.3) is the upper-level objective function.

The rib distances and stringer intervals are adjusted in such a way that the local buckling of the skin must not occur before the wing panel reaches its global buckling critical load.

The layout of ribs is determined by optimizing the length of each panel, which mainly affects global buckling resistance. Wing panels are converted into equivalent orthotropic plates that simulate their mechanical features by global stiffness parameters (superposition of skin's stiffness and stringers' stiffness). The global buckling critical loads of equivalent anisotropic plates are defined by Rayleigh-Ritz energy method. Both static strength and global buckling resistance are predicted using these equivalent models. A surrogate model based on artificial

neural network is proposed to represent the relationship between structural parameters and local buckling load.

At the lower level, a parametric modeling program is employed to provide static analysis, which could extract in-plane loads on each wing panel. Then these equivalent panels are optimized to increase their efficiencies. In this process, layout parameters are converted into continuous stiffness and global dimensions of these panels. Since the predictions of failure coefficients are based on analytical methods, the panel optimization does not rely on FE model, which could not be an expensive running. Finally, optimized results are transformed back into layout parameters to adjust the positions of components. Rib distance is determined by the length of each wing panel.

When subsystem optimization is completed, the deflection of the sized structure is compared with target limit. If the constraint is violated, the structural stiffness is needed to be increased.

The two-level system is coded into a nested loop mode. Since FE model is only utilized for static analysis once in every circulation, and wing panel optimization is mainly based on approximated mathematical models, the whole process would not consume many computational resources. The example demonstrates that this strategy could converge in a few iterations.

For the preliminary layout design, only main global features are considered in this paper. Aerodynamic load causes bending and twist deformation on wing structure, which induces compressive/tensile and shearing loads in wing panels. The strains are constrained by allowable compressive and shearing strains.

To verify the developed approach the wing structure layout of a certain large transport aircraft was optimized. The wingspan is 17 m, and the maximum takeoff weight is supposed to be 62 000 kg. The structure is required to withstand 2.5g overloading without any strength or buckle failure, meanwhile, wing tip deflection should not surpass 10% of the wingspan. Unidirectional carbon fiber T300 is used as the material for wing panels.

The optimization processes of wing panels are based on the sequential quadratic method, which is integrated in SOL200 procedure of general software NASTRAN; the entire two-level system is coded using Patran Command Language (PCL) program, which could be integrated into PATRAN software as a secondary development module.

The algorithm was applied to two different initial layouts of the wing. The number of iterations till the convergence was 17 and 26 for these initial layouts. It shows that this method gets converged rapidly, and the result is insensitive to initial layout. The optimized configurations and structural parameters for different initial structures are basically in accordance with each other, which demonstrates a satisfied global optimization capability of this method.

The developed methodology is a good and effective approach to obtain a preliminary optimal design layout of a wing structure. However, it cannot be used for other types of

structures. Moreover, the optimization parameters for the skin and stringers are only their dimensions and thicknesses. There is now information about how the laminate stacking sequences were chosen and optimized. The blending problem is not solved in the paper also.

An interesting approach for a two-level multi-objective optimization of a composite wing was proposed by Y. Hailian and Y. Xiongqing [19]. The objectives of the study was minimization of the manufacturing cost and the weight of the wing structure.

At the upper (system) level the layout of the structure is optimized. A set of parameters, which represent the external shape and the structural layout are defined. The parameters for the wing external shape definition are the area, the aspect ratio, the taper ratio, the sweep angle, and the airfoil. The parameters for the structural layout include the number of spars, the number of ribs, and the locations of the front spar and the rear spar. The parameters for structural layout are taken as the design variables and are changed during layout optimization. The task of structural layout optimization is to find the Pareto optimal set for the wing structural layouts with minimum weight and cost. This multi-objective optimization problem is solved by the non-dominated sorting genetic algorithm (NSGA-II [20]), which is build-in to the parametric optimization software Dassault iSIGHT.

At the lower level a structural dimensions optimization is carried out. Its task is to find the dimensions of structural elements with minimum weight under the constraints of the allowable stress or strain of all materials, and the structural deformation and bucking criteria. In addition, the composite failure criterion is also used as a constraint for composite wing design. The structural dimension optimization is implemented with software MSC Patran/Nastran.

When the structural layout parameters are defined, a CAD model of the wing structure is automatically generated by a Microsoft Visual Basic (VB) routine that has an interface with Computer Graphics Aided Three Dimensional Application (CATIA) software.

Then the generated CAD model of the wing box with the format of initial graphics exchange standard (IGES) is imported to the MSC Patran software.

By using PCL of MSC Patran software, a finite element model for the wing box is generated automatically based on the imported CAD model. The initial value of the structural dimensions, the information of material, the degree of freedom (DOF) constraints, and the aerodynamic loads are defined using PCL in the finite element model. The parameters for structural dimensions are the thickness of the skin, the thicknesses of the spar webs and the ribs, and the sectional area of the bars. The initial dimensions of the structure are given based on the previous experience.

All the steps can be integrated using the parametric optimization software Dassault iSIGHT. The overall process is executed automatically.

A structural design optimization for the composite wing of an unmanned aerial vehicle (UAV) is used as an example to verify the applicability of the proposed method. The takeoff weight of the UAV is 1 580 kg. The external shape of the wing is defined as follows: the area

of the wing is 6.5 m2, the aspect ratio is 10.0, the taper ratio is 0.5, and the sweep angle is 12°. The Carbon/Epoxy T300/5208 material is used for all parts of the composite wing. The material ply orientations are limited to 0°, ±45°, and 90°. The plies are symmetric for all parts.

The design variables in dimension optimization are the thickness of skin and web. The wing structure is divided into three sections along wingspan. Each section consists of the upper skin, lower skin, spars and ribs. The thicknesses of the parts (upper skin, lower skin, spar webs and rib webs) are identical for same section but are variable for different sections. Each section has four design variables describing the thickness of each ply orientation. The total number of variables is 60 when the wing has two spars.

The described above approach is very interesting and promising from the point of view of the modern design approach. It is fully automated. It uses modern and common CAD and FEA software and programmed internal routines, which generate CAD and FE models. It uses GA as the upper-level optimization method.

However, it uses MSC Nastran SOL200 for optimization of structural dimensions (thicknesses of composite skin, webs, and ribs – discrete parameters) at the lower-level. Since MSC Nastran SOL200 does not allow to operate with discrete parameters, this approach is not fully correct and effective. Moreover, the ply orientation angles 0°, ±45°, and 90° are used only, what limits the optimization domain and suboptimal designs could be found only. It is better to use GA or another discrete optimization method at the lower level.

O. Seresta et al. [21] proposed a two-level optimization algorithm, which is based on a GA, which operates by so-called guides. A guide is a basic template laminate stacking sequence (defined by a vector of ply angles chosen from a discrete set of angles) that is applicable to all the designated panels, which the entire optimized structure is divided into. From this guide design, a certain number of contiguous plies are kept to represent a particular panel. This ensures complete blending.

The design optimization problem is formulated as a minimum weight design of a composite wing structure subjected to the maximum global deflection, global blending, local strength failure, local ply contiguity (successive plies of same fiber orientation), and local panel buckling constraints. The design variables are the stacking sequence of the guide, and the number of layers to be kept for each panel. The global blending constraint is satisfied by using guide-based design. The individual panel stacking sequence for the *i*-th panel is defined uniquely as the outer (for outer blending), or inner (for inner blending) number of plies of the guide stacking sequence.

The first part of the individuals (stacking sequence of guide) of the initial population is generated randomly. The second part or string is initially assumed to be equal to the number of plies in the stacking sequence of the guide. Thus, at the start of optimization process, all the local panels are assumed to be of uniform thickness equal to that of the guide.

Every individual (or guide design) in the population is assigned a fitness value based on the overall performance of the guide, which is measured by the total weight of the structure (dictated by the number of layers in each panel presented in the second string) and the value of

the constraints. If the constraints are satisfied then the fitness value is the total weight of the structure, otherwise, the fitness is the total weight of the structure plus a penalty for the constraint violation.

The fitness evaluation of an individual (or a guide design) requires computation of the structural weight as well as the constraint violations or margins, which are obtained via global and local analyses. Based on the number of plies that are to be kept from the guide, the laminate stacking sequence is assigned to all the panels, and a global FE analysis is performed that gives the in-plane loads acting on each of the panels. The maximum deflection constraint is checked and if violated, a penalty is applied to the fitness evaluation. Then for each panel the local strength constraint and the local buckling constraint is computed, and the constraint margin is calculated. The local strength and buckling analyses are performed using classical laminated plate theory. Depending on the constraint margin value, a decision is made either to add a ply (if the constraint margin is negative i.e., panel has failed) or remove a ply (if the constraint margin is positive i.e., panel is safe) while keeping the load for the panel constant. That is, no new expensive global FE analysis is done to compute the new in-plane loads due to change in local stiffness and the local panel analyses are repeated to check the local constraints. The decision to remove a ply is only made if the global deflection constraint is satisfied. If the constraint margin after removing a ply is still positive, then all the calculations pertaining to the local panel are done based on the new number of plies. Otherwise, calculations are performed based on the previous value before removing a ply.

This stripping or adding plies at the local level is referred to as local improvement. The number of layers that is to be kept from the guide for each of local panels due to local improvement is stored for future reference. While improving the current pool of designs at the local level, the obvious disadvantage of the method is that as soon as the number of layers is changed at the local level, the previously performed global FE analysis is no longer valid. In order to prevent the local in-plane panel loads from jumping severely from one global FE analysis to another, the local improvement performed at the local level is restricted changing at most one layer per panel.

In order to speed up the computational effort, evaluation of the performance of the individual designs of the populations, which require running the complete FE wing analysis, is carried out using a node cluster. A simple master–slave parallel code is implemented for this work. The master process generates and runs the GA code while distributing guide designs to slave processes for analysis in a lock step message-passing phase. For a population size of 400, 25 slave processes were used distributing 16 guides to each process to be analyzed in a given iteration.

The developed algorithm was applied for a simple composite wing box design optimization problem originally presented by Liu et al. [22]. The wing structure model used is a straight rectangular wing box with the dimensions 3543.3×2240.3×381 mm. The top surface and the bottom surface layers are the targeted design panels. The possible ply orientations for the design panels are 0°, ±45°, and 90°, while all other panels are fixed to the design $[\pm 45°_{11}]_S$. All the panels are modeled using membrane elements. The root of the wing is fixed, and the

load is applied at the free end. The total numbers of panels at the top and bottom layer are 9 each. The wing box is made of graphite-epoxy T300/N5208.

A single upward loading case is considered in this paper. Tip loads at the free end of the wing simulate the upward bending loads. The upward lift force acting is modeled by four concentrated loads. This set of loads induces both upward bending and twisting of the wing box.

Two different guides are chosen to represent the top and the bottom skin laminates, respectively. Global finite element analysis is performed using the commercial finite element package GENESIS. An interface module is built to communicate between GENESIS and the locally developed GA code. The optimal designs reported are obtained after 300 GA generations. Optimal designs for both outer and inner blending are reported.

The upper nine panels are mainly subjected to compressive load and the lower nine panels are under tensile load. Buckling, strength, and ply contiguity constraint is applied to upper nine panels only. No ply contiguity and buckling constraint is imposed on lower nine panels. At this stage no deflection constraint is imposed on the optimization problem. The initial population was of 400 individuals. The maximum length of chromosome was set to 150 and 50. The probability of crossover and mutation were set to 1.0 and 0.0 correspondingly.

There are three main advantages of the developed algorithm. The first one is in that the optimized structure is fully blended because of guide-based design. The second is decreasing of computational effort required for FE analysis of the global model, since this analysis is performed only once for each new guide. The third advantage is parallelization of global model FE analyses being performed for different guides.

However, there exist several disadvantages in the algorithm. It is not effective to use the predefined stacking sequences of guides at the global level for calculating the load distribution, since they are narrowing the optimization domain. Moreover, in each initial design all the panels have the same thickness, which complies with the full stacking sequence of a corresponding guide. Thus, to be consistent with the global level in terms of load distribution the local level designs of the panels cannot be far from this thickness. This is narrowing the optimization domain also (e.g., for long narrow structures it is better to gradually reduce the thickness of laminates from the root to the tip). Since the strength and buckling constraint margins are evaluated at the local level, there exist a risk of violating these margins at the global level when the panels' thicknesses were reduced and consequently the internal loads were redistributed. However, the algorithm does not propose any additional checks at the global level. The stresses and buckling load factors at the lower level are calculated according to the classical laminate plate theory. This limits the algorithm application to the structures with a small curvature.

D.B. Adams et al. [23] proposed a variation of the algorithm of O. Seresta et al. [21] described above. It is based on the guide design methodology. Each laminate is encoded for use in the GA. Integer values from zero to seven represent the orientation of each ply. The positive integers map to orientation angles 0°, 15°, 30°, 45°, 60°, 75°, and 90° from one to seven, respectively. The zero represents an empty ply. Successive occurrences of a gene map to the

orientation with alternating plus and minus signs, starting from the center of a symmetric laminate. For example, the first occurrence of a 2 encoding maps to 15°, the second 2 maps to −15°, and so on. This is how a balanced laminate is encoded. Only a half of the orientations is required to be encoded, since symmetrical laminates are covered in the study.

To define a particular structure design, the guide is accompanied by the second string of integers. The string contains as many integers as many panels the structure is divided into. Each integer corresponds to the number of layers to be removed from the guide to obtain a particular panel design.

The algorithm is divided into three main stages. The first one is so-called global/local iterative process which performs a FE analysis of a global model and evaluates the fitness value of a particular guide. The fitness value of a guide is defined as

$$F = -W(P + 1), \tag{1.4}$$

where W is the total weight of the structure and P is the total penalty applied to the structure for being unbalanced.

If, however, any of the panels in the structure fail under their local loading requirements an additional multiplicative penalty is applied to the fitness calculation making failed designs highly undesirable.

The local level optimizations take a single panel with the given loads returned from the most recent global analysis and determine the maximum number of layers, which can be removed from the initial subsequence of a guide that can satisfy the given loading requirements. The local algorithm removes or adds a layer at a time and after a local FE analysis is performed. When the optimal designs of the panels are found the global FE analysis is performed again.

A parallel GA, using the local/global method to evaluate guide designs, is constructed where the determination of fitness for a member of the population is assigned to a worker node during the analysis phase of each generation while the GA itself is controlled by a single master process. The evaluation of a single guide begins with the global analysis of a completely thick design: each panel in the structure is assumed to use every possible layer from the given guide.

The authors proof the best design found within the first stage does not correspond to the global optimum. Therefore, they propose two more stages to improve the guides from the last iteration of the first stage.

Each design of the structure is encoded as an array of integers representing the number of layers removed from the guide for each panel in the structure. The authors propose to interpret each guide found at the previous stage as an acyclic directed graph structure (with a source node) of all possible designs (encodings) derivable from a given guide by manipulating the corresponding array of panel descriptors. The source for a four-panel structure would be the fully thick design designated by the list (0,0,0,0), where no ply layers are removed from the guide for any panel. The position of a number in the list corresponds to the design of a specific panel. The children of a specific node in the digraph correspond to all possible designs

obtainable by removing a single ply layer from each panel in turn. For this example, the four children of the source node are the designs (1,0,0,0) (0,1,0,0) (0,0,1,0) and (0,0,0,1).

Thus, the authors propose to make an estimation of the computational budget of each guide in the second stage. The guide designs, which require too much computational effort to find the design with the minimum possible thicknesses of the panels are discarded from the further search. The others taken as the starting nodes of acyclic directed graph structures are subjected to a bounded implicit enumeration algorithm within the third stage.

The developed algorithm was applied to optimization of a simple wing box structure. It is unswept, untapered and rectangular with dimensions 3543.3×2240.3×381 mm. The spars and the ribs divide the top and bottom skin of the wing box into four panels of equal size. The top and bottom skin panels are modeled using membrane elements. Only the top skin panels are being optimized, all other sections are fixed to the design $[\pm 45_5/45]_s$. The root of the wing is fixed and load is applied at the free end. Global finite element analysis is performed using the commercial finite element package GENESIS. The upward lift force acting on the wing tip is modeled by three concentrated loads.

This problem was solved on a 200 node (400 processor) 1.4 GHz dual Opteron cluster using an MPI (message passing interface) based Fortran 90 code. Because of a small size of the problem 51 processors were used only. The average run time for a GA using a population size of 50 running for 200 generations was about 12 hours (612 node hours). For comparison this problem was examined using the algorithm by Seresta et al. [21]. For a run of 200 generations of the GA, Seresta's algorithm discovered in 50 generations a solution that it could not improve after an additional 150 generations. The weight of the upper skin for this design was 12.371 kg. The algorithm by D.B. Adams et al. discovered a different solution of equal weight in one to four generations of the GA and a lighter design was discovered after 100 iterations. This lighter design had the upper skin weight of 12.018 kg.

The advantages of the algorithm are the same as were discussed for the above-described algorithm by Seresta et al. [21]. The extra advantage is in the third stage direct search, which allows to carefully discover all the possible designs for the guides chosen at the first stage. The application to the test problem proofs it by finding a lighter design. From the other hand it can be taken as a disadvantage of the algorithm, since it requires huge computational resources. The other disadvantages are similar to those of the algorithm by Seresta et al. [21].

The examined above algorithms propose different approaches to the composite structures optimization problem. They have their advantages and drawbacks. The main disadvantages are connected with:

- limiting the application of the algorithm to a certain type of structures, e.g., wing,
- using a special FE method, which is not available in a common FEA software,
- narrowing the optimization domain by fixing thickness or stacking sequence (or limiting it to a small number of angles, e.g., 0°, ±45°, and 90°) of the optimized sub-structures,

- not using (or using in an ineffective way) the coordination between the upper and lower levels of problem decomposition,
- lacking universalism.

The most advanced and promising algorithms use a GA as the only or one of the optimization methods. This choice can be supported by the reviews [10], [24] – [26]. According to them GAs obtain the highest rankings in comparison to the other algorithms applied to composite structures optimization.

Finally, based on the above made review an image of a modern optimization methodology for composite structures can be created. The basic aspects of the methodology could be the next:

1) it should be universal enough to be applicable to a wide class of composite structures,
2) it should be able to use (integrate) the modern CAD and FEA software for building and analyzing FE models of the optimized structures,
3) it should decompose the optimization problems into several levels,
4) several different optimization methods should be used at different levels,
5) because the nature of the optimization problems supposes a large number of variables and several local optima a stochastic optimization method (e.g., GA) should be applied at one of the levels,
6) means to minimize the number of variables should be taken,
7) to accelerate the algorithm a parallelization of calculations should be provided.

2 Formulation of the optimization problem

The above given state-of-art review allows defining the optimization problem in detail, taking into account specifics of the airframe composite structures.

When we think of composite airframe components, first what is emerged in our mind are the monocoque or semi-monocoque structures (see Fig. 2.1). That means, in general, they consist of almost the same structural elements and their design approach is very similar. Thus, it is possible to create an optimization methodology, which will be in general applicable for optimization of almost any airframe component (wing, fuselage, etc.).

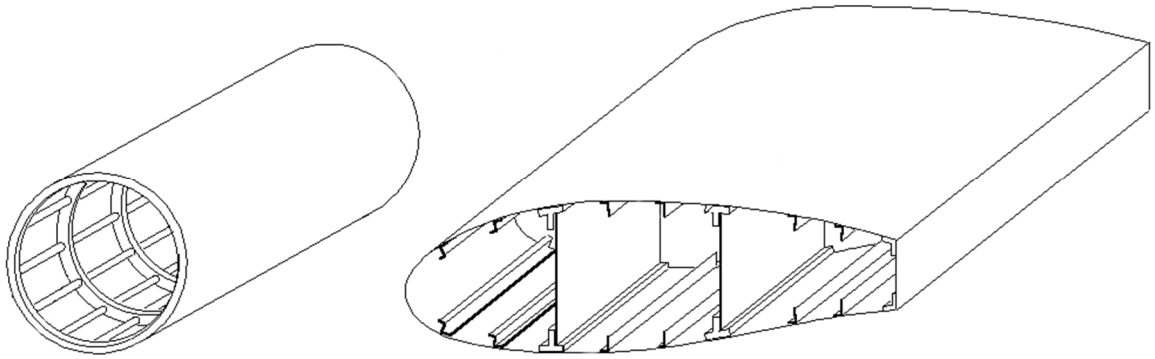

Fig. 2.1 Semi-monocoque structures

The airframe structures are complex assemblies, and each assembly unit consists of many structural elements. The main of them are spars, frames or ribs, stringers, and skin. Moreover, every laminate itself may be a complex assembly of plies, which may be in general made of different materials and have different fibers' orientation (see Fig. 2.2). These circumstances are the reason that the number of design variables necessary for optimization of a large-scale composite structure counts a few thousands [9].

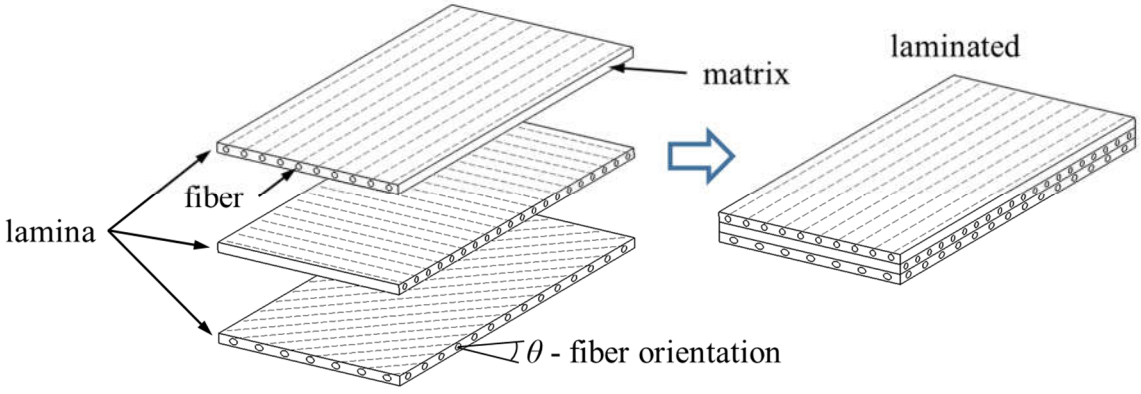

Fig. 2.2 Laminate "assembly"

As it was discussed above, in such a case, it is not effective to do direct optimization. Therefore, the optimization procedure of a large-scale composite structure is decomposed onto levels. Usually there are two main levels: upper-level and lower-level optimization.

At the upper level, the global parameters, which significantly influence the behavior of the entire structure, should be chosen and optimized. In the most of cases the internal load distribution and in some cases the shape of the structure are optimized at the upper level. The local parameters of the structural units and elements, laminate thicknesses and stacking sequences are not taken into account. Additional optimization objectives can be different for different structures. For example, if the aerodynamic shape is fixed, the aerodynamic surfaces can be optimized for maximum flatter speed and minimum deflection and twist angle. The deflection or twist angle also can serve as constrains. The constraints can be technological and design ones also. For example, from the technological point of view the panels' sizes may be limited by the manufacturing tools and facilities. Also, the continuity of the structure should be provided as it is described in the beginning of the previous section.

At the lower level, the optimization process is broken into several local sub-problems. The sub-problems are characterized with local parameters and constraints, which have small influence on the entire structure. These parameters and constraints could be unique for each sub-problem. Usually, the lower-level optimization deals with the thickness and stacking sequence of the laminate, plies' orientation, and the geometrical parameters of the structural elements. The optimization objective at this level is mainly the structural weight minimization limited at least by strength and stability constrains. The loading is kept constant. Additional objectives may be maximization of service life, maximization of panel dimensions, maximization of crashworthiness and impact value, etc. In general, the local optimization problem may be written as follows:

$$\begin{gathered} W(\bar{X}) \to min, \\ F_i(\bar{X}) \to min(max), \\ \min_j RF_j(\bar{X}) \geq 1, \end{gathered} \tag{2.1}$$

where W - weight of the local substructure; $\bar{X}$ – vector of variables (number of plies, stacking sequence, geometric dimensions, etc.); F_i – other objective functions; RF_j – reserve factors of different failure modes, including buckling ones.

The upper and lower levels are coupled with help of coordination algorithm. Authors in [1] and [2] propose an upper-level coordination constraint, which require an approximation of the lower-level objective function and its complex and expensive sensitivity analysis. This algorithm is the bottle neck of such an approach. In case of composite structure such an analysis becomes even more complex and expensive because of large number of lower-level variables. Another approach is to make very small changes at the local level, e.g., to remove or add at most one ply per panel for one step of local level optimization [27], which allows to stay not far from the upper-level stiffness and loads distribution. However, in case of thick laminates the effectiveness of such an approach can be low.

Thus, the above noted discussion makes evident that the optimization of a large-scale composite structure is a complex multi-stage, multi-disciplinary, multi-objective, and multi-parametric task.

3 Objectives of the thesis

Within the present study it is planned to **develop an optimization methodology for thin-walled composite structures** which, in general, is divided in several sub-structures with different stiffness (see Fig. 2.1). Each sub-structure may, in general, consist of several skin panels of different thicknesses, spars and stringers and separated from the neighboring sub-structures with ribs or frames. The structure may be loaded with aerodynamic, weight, inertial forces and in some cases with internal pressure. Several load cases should be considered. The cross section of the structure may be single-cell, multi-cell, open or combined.

Each structural element (skin, stringer, spar web, spar cap, etc.) may have different thickness and stacking sequence. Concerning materials, the symmetrical orthotropic laminates are considered only. In the most general case, the orthotropic laminate of each structural element may have such a stacking sequence $[0°, 90°, \pm\theta_1, \pm\theta_2, \dots, \pm\theta_k]_{ns}$.

In general, the optimization parameters will be the following:

- thickness and stacking sequence of each structural element being optimized,
- angles $\pm\theta_1$, $\pm\theta_2$,…, $\pm\theta_n$ for each structural element, in which they are used.

The input data for the optimization will be:

- baseline geometrical concept of the structure (means outer shape, position of spars, stringers and ribs, dimensions of the skin panels),
- materials and their properties,
- load cases.

The optimization objectives should be, in general, the following:

- weight minimization of the empty structure,
- deflection and twist angle minimization (these parameters could be used as constrains), etc.

The optimization constrains should be the next:

- static strength, local and global stability of the structure,
- maximum deflection and twist angle of the whole structure,
- maximum deflection of the skin panels,
- maximum thickness of the structural elements,
- blending rules.

At least **one application of the methodology should be shown** within the present work. The effectiveness and robustness of the methodology in comparison to the existing ones should be shown also.

4 Optimization methodology

The optimization methodology comprises the problem decomposition approach, optimization methods used at different levels, the coordination procedure required to interconnect the different levels and the analysis methods used at these levels.

The optimization method used at the upper level depends on the defined objectives, however in general it is an aeroelastic problem, which can be solved by joint aerostructural methods as in [4], [18] or [28].

Since the present work is mostly focused on the structural problems, it is not planned to develop any aerostructural method within this thesis. Only structural optimization is developed here. Anyone can adopt and connect an appropriate aeroelasticity optimization method, which uses the upper-level structural parameters discussed below.

The upper and lower levels are usually connected to each other via stiffness of the structure. The structure is usually divided onto sub-structures, which have constant stiffness within their limits. Thus, the structural cross-sectional stiffness (axial, bending, torsional, etc.) of each sub-structure may be taken as an optimization variable at the upper level. For example, the authors in [4] propose expressing the bending stiffness submatrix of a laminated panel through integral parameters (1.1). Then the cross-sectional stiffnesses can be expressed through these parameters. At the lower level these parameters can be connected to the thickness and the orientation angles of the laminate's stacking sequence.

The lower-level optimization problem, when the laminate thickness is one of the variables, has discrete character, moreover the equations of the laminates' mechanics are quite bulky and complex.

Thus, the classical optimization methods, e.g., the Lagrange multiplier method and gradient ones, etc., where differentiation and integration are required are not appropriate for this problem solution. Also, according to [26] these methods become ineffective when the number of variables is more than 10, moreover, these methods are not good for complex optimization space with several local optima because of possible premature convergence.

These disadvantages are not inherent to the stochastic methods like GA [29] and swarm based methods [30] – [32]. However, the stochastic methods are quite expensive because of many calculations of objective functions. This disadvantage may be eliminated by above noted parallelization.

The GAs have one more advantage in case of composite materials – they are very convenient for coding stacking sequences and dealing with discrete fenomena. An example of coding of a laminate stacking sequence is shown in Fig. 4.1. Since only symmetric laminates are discussed in this study, a half of the sequence is coded only.

Moreover, the stochastic methods are very simple in sense of programming.

According to [26] and [33] GAs obtain the highest rankings in comparison to the other algorithms applied to composite structures optimization.

Fig. 4.1 Stacking sequence coding of a laminate

Hybrid methods, e.g. [34] – [37], can be even more effective.

In general, the multilevel optimization procedure may consist of blocks shown in Fig. 4.2. Within the upper level the weight of the structure and the internal load distribution are optimized. The local parameters of the sub-structures and their parts (laminate thicknesses and stacking sequences) are not considered. Additional optimization objectives can be different for different problems. For example, if the aerodynamic shape is fixed, the aerodynamic surfaces can be optimized for maximum flatter speed and minimum deflection and twist angle. The deflection or twist angle also can serve as constraints. The constraints can be manufacturing and design ones, also [37]. The continuity of the structure should be provided also.

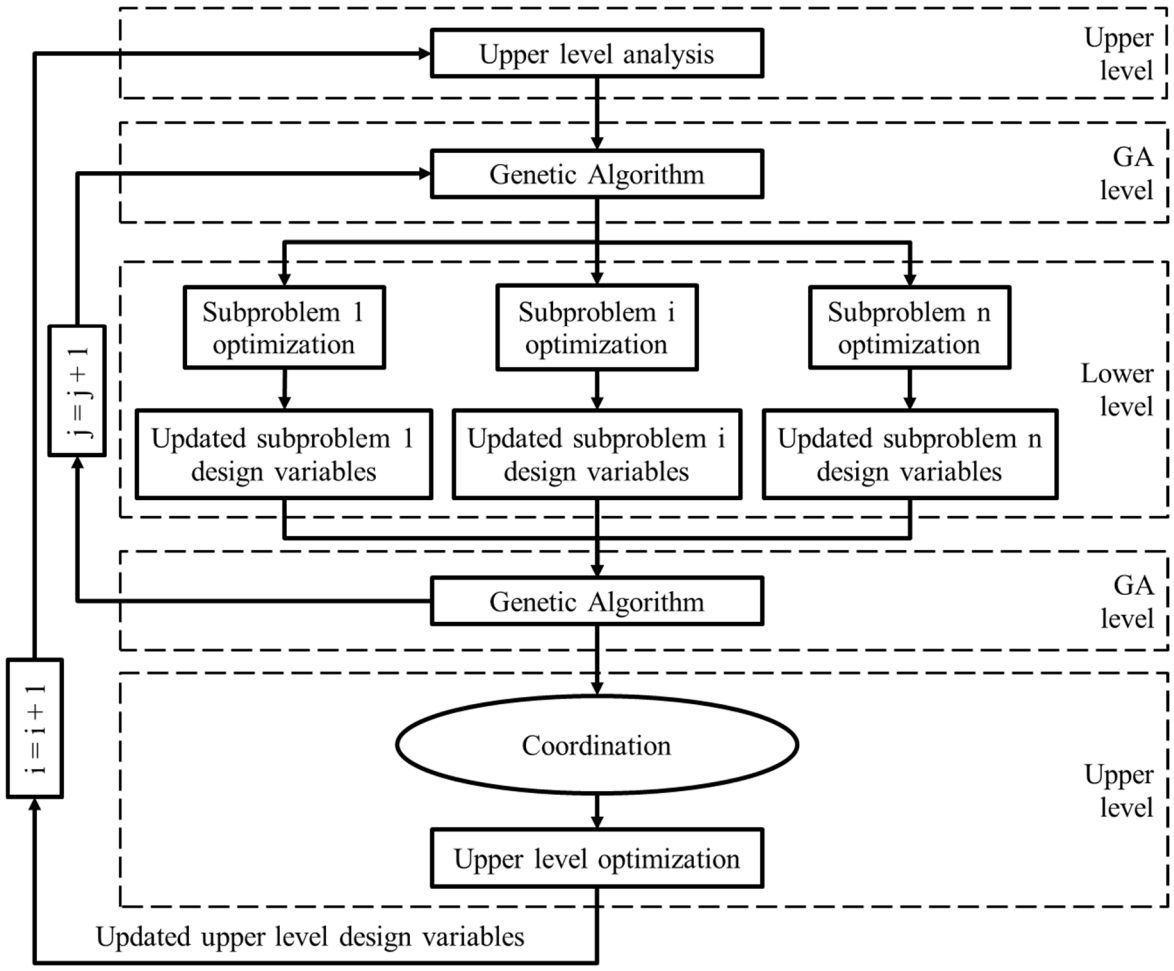

Fig. 4.2 Optimization flowchart

At the GA level the optimization deals with stacking sequences of laminates of the structure. A genetic algorithm is used here to optimize the stacking sequences. The optimization objective of this level is to minimize the difference between the cross-sectional stiffnesses calculated at the upper and lower levels. The optimization constraints are the maximum and

minimum thicknesses of laminates and the orientation angles of plies (the set of angles is set by a designer according to manufacturing constraints).

At the lower level the optimization deals with the thickness of the laminate. The optimization objective of this level is to minimize the difference between the values of upper-level variables calculated at the upper and lower levels. The optimization constrains are at least the strength and buckling of the local sub-structures. The loading is taken from the upper level and kept constant.

The optimization algorithm starts with the upper-level analysis of the initial global design of the structure, which provides the algorithm with the initial global responses and parameters.

This information is passed to the GA level. At this level a set of guiding stacking sequences (guides) is generated (in detail see Subsection 4.2). A guide represents a laminate of maximum allowed thickness. The ply angles of the guide are supposed to be fixed. The guides' stacking sequence is supposed to be the same through the structure or its part (the designer decides). Only thickness can be varied through the structure. The GA deals with the guides. Its goal is to find the best guide, which will provide the best correlation between the upper and the lower levels.

The guides are sent to the lower-level sub-problems. Then the optimization of each lower-level sub-problem is performed. During this step the thickness of each local sub-structure should be optimized in such a way, that the in-plane stiffness of the sub-structure corresponds as much as possible to that found during the previous upper-level analysis. At the same time the local constraints should not be violated. The information from the lower-level optimization is passed to the GA level. Then the best guides found at the GA level are passed to the coordination procedure. In the last step the upper-level optimization follows. The convergence of entire optimization algorithm will occur, when the upper-level objective function is optimized, while the upper and lower-level constraints are satisfied.

In further subsections a particular realization of the upper-, GA- and lower-level algorithms is proposed and described in detail.

4.1 Upper level

At the upper level the entire structure characteristics and responses are in focus, such as weight, deflections, twisting angle, global buckling, minimum flutter speed, etc. All of them depend on the material and stiffness distribution along the structure. Thus, the most important global parameters of the structure correspond to its cross-sectional stiffnesses (axial, bending and torsional). Since they are not independent from each other and depend on the geometry and materials of a cross-section, it will not be correct to vary them directly and independently. There should be found independent parameters, which influence these stiffnesses. Since at the upper level it is not important what kind of geometry and materials will be applied at the lower level, the structure at the upper level can be modeled simplistically. If the structure is simple a conventional mathematical model from the structural mechanics or a surrogate model can be used, i.e., panel, beam, or a thin-walled bar, e.g., [4]. When the structure is complex, FE

modeling can be used. FE model is more appropriate for modern engineering, where the design process starts from the global FE model and protrudes to the local ones.

The number of parameters describing the upper-level model should be as less as possible in order to simplify and accelerate the optimization at this level. For example, as it was discussed in Section 1, the authors in [4] and [38] proposed to use lamination parameters for varying bending stiffness sub-matrix of laminated panels (1.1). Since the sub-matrix D is a function of only 4 lamination parameters and there is no need to use orientation angles and number of plies, it is very convenient for global optimization. However, if the structure is loaded not only with bending, but with tension and torsion the membrane stiffness matrix A is important also (for orthotropic laminates the coupling sub-matrix B is empty). Thus, the total number of lamination parameters will be 2 times more, than in case with bending. In this case the authors in [38] propose to use a smeared stiffness approach for defining composites at the upper level. With such an approach the membrane and bending stiffness sub-matrices are related to each other as follows:

$$D = A\frac{h^2}{12}, \tag{4.1}$$

where h - total thickness of a laminate.

In case of orthotropic composites only 2 lamination parameters are needed to define these two sub-matrices.

Of course, in a complex structure, where several structural elements with different stiffnesses are in the cross-section, each element has its own A and D sub-matrices. Thus, the total number of upper-level parameters will be equal to number of structural elements (only those having unequal laminate stiffness matrices) multiplied by 2.

However, the lamination parameters are not independent quantities, and their physical meaning is questionable. It is more convenient and sensible to use some parameters, which are independent and has definite physical meaning. As it was deiscussed above, the parameters, which can be used for the coordination between the upper and lower levels are the bending and torsional stiffnesses of the cross-section (see [2]). The authors propose the next lower-level objective function:

$$f = \left[\frac{EJ_z - EJ_z^*}{EJ_z^*}\right]^2 + \left[\frac{EJ_y - EJ_y^*}{EJ_y^*}\right]^2 + \left[\frac{GJ - GJ^*}{GJ^*}\right]^2, \tag{4.2}$$

where EJ_z , EJ_y – bending cross-sectional stiffnesses, GJ – torsional cross-sectional stiffness. The star symbol (*) denotes the stiffness calculated at the upper level using the global model parameters.

The stiffnesses depend on the material and geometry of the cross-section. In turns, for the thin-walled structures the geometry is defined by the thickness and the shape of the cross-section. For example, the bending stiffness of a simple cross-section (see Fig. 4.3), which

consists of 4 composite panels with different thickness and stacking sequences [39] can be written as follows:

$$EJ_z = EJ_{zup} + 2EJ_{zs} + EJ_{zl}\,, \tag{4.3}$$

where $EJ_{zup}, EJ_{zs}, EJ_{zl}$ – bending stiffnesses of upper, side and lower panels correspondingly, which can be defined by the next formula:

$$EJ_{zi} = h_i E_{xi} \int_{s_i} y^2 ds = (hE_x)_i \bar{J}_{zi}, \tag{4.4}$$

where $i = (\overline{up, s, l})$, h_i and E_{xi} – thickness and longitudinal elastic modulus of $i-th$ panel correspondingly, s – curvilinear coordinate along the cross-sectional contour, $\bar{J}_{zi} = \int_{s_i} y^2 ds$.

The equation (4.3) can be rewritten taking into account (4.4):

$$EJ_z = (hE_x)_{up}\bar{J}_{zup} + 2(hE_x)_s\bar{J}_{zs} + (hE_x)_l\bar{J}_{zl}. \tag{4.5}$$

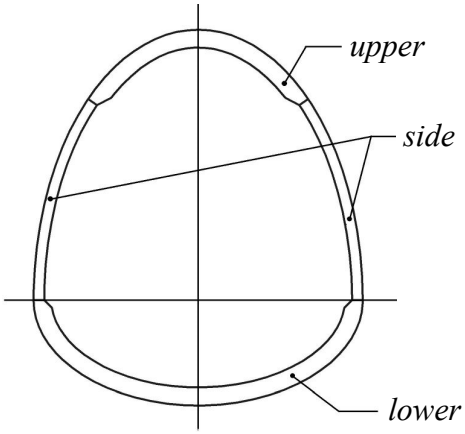

Fig. 4.3 Simple cross-section split into 4 panels

The stiffness EJ_y can be expressed in a similar way.

When the geometry is fixed, the quantities $\bar{J}_{zi}$ are constant. In that case the bending stiffnesses depend on $(hE_x)_i$ products only. Also, it can be shown, the torsional stiffness GJ depends on $\left(hG_{xy}\right)_i$ products only (G_{xy} – in-plane shear modulus of the panel).

4.1.1 Definition of laminate stiffness matrix

From the other hand the membrane stiffness sub-matrix A of each panel can be defined with help of lamination parameters in the same way as it was done in [38]. For orthotropic laminate the membrane stiffness sub-matrix can be formulated as:

$$A = \begin{bmatrix} A_{11} & A_{12} & 0 \\ \vdots & A_{22} & 0 \\ sym & \dots & A_{66} \end{bmatrix}. \tag{4.6}$$

The nonzero members of the matrix can be written through the lamination parameters as follows:

$$\begin{Bmatrix} A_{11} \\ A_{12} \\ A_{22} \\ A_{66} \end{Bmatrix} = h \begin{bmatrix} 1 & \bar{\xi}_3 & \bar{\xi}_1 & 0 \\ 0 & 0 & -\bar{\xi}_1 & 1 \\ 1 & -\bar{\xi}_3 & \bar{\xi}_1 & 0 \\ 0.5 & 0 & -\bar{\xi}_1 & 0.5 \end{bmatrix} \begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{Bmatrix}, \tag{4.7}$$

where $\bar{\xi}_1 = \frac{\xi_1}{h}$ and $\bar{\xi}_3 = \frac{\xi_3}{h}$ – lamination parameters, the constants U_i depend on the lamina stiffness matrix Q:

$$\begin{aligned} U_1 &= \frac{3Q_{11} + 3Q_{22} + 2Q_{12} + 4Q_{66}}{8}, \\ U_2 &= \frac{Q_{11} - Q_{22}}{2}, \\ U_3 &= \frac{Q_{11} + Q_{22} - 2Q_{12} - 4Q_{66}}{8}, \\ U_4 &= \frac{Q_{11} + Q_{22} + 6Q_{12} - 4Q_{66}}{8}. \end{aligned} \tag{4.8}$$

The (4.7) can be rewritten as follows:

$$\begin{aligned} A_{11} &= h\left(U_1 + U_2\bar{\xi}_3 + U_3\bar{\xi}_1\right), \\ A_{12} &= h\left(-U_3\bar{\xi}_1 + U_4\right), \\ A_{22} &= h\left(U_1 - U_2\bar{\xi}_3 + U_3\bar{\xi}_1\right), \\ A_{66} &= h\left(\frac{U_1}{2} - U_3\bar{\xi}_1 - \frac{U_4}{2}\right). \end{aligned} \tag{4.9}$$

Let's introduce a term:

$$k = \frac{E_x}{E_y} = \frac{A_{11}}{A_{22}}. \tag{4.10}$$

Using the term k it is possible to rewrite the last three equations from (4.9) as the following:

$$\begin{aligned} \frac{A_{12}}{h} &= -\frac{A_{11}}{h}\frac{k+1}{2k} + U_1 + U_4, \\ \frac{A_{22}}{h} &= \frac{A_{11}}{h}\frac{1}{k}, \\ \frac{A_{66}}{h} &= -\frac{1}{2}\left(\frac{A_{11}}{h}\frac{k+1}{k} - 3U_1 + U_4\right). \end{aligned} \tag{4.11}$$

Thus, it can be seen, the entire extensional sub-matrix normalized with the thickness of the panel can be defined by $\frac{A_{11}}{h}$ and k terms only.

From the other hand the $\frac{A_{11}}{h}$ term can be expressed via engineering constants:

$$\frac{A_{11}}{h} = \frac{E_x}{1 - \mu_{xy}\mu_{yx}}, \tag{4.12}$$

where E_x – longitudinal elastic modulus of the panel, $\mu_{xy} = \frac{A_{12}}{A_{22}}$ and $\mu_{yx} = \frac{A_{12}}{A_{11}}$ – Poisson's coefficients of the panel.

With help of (4.11) the expression (4.12) can be transformed to the next quadratic equation, where $\bar{A}_{11} = \frac{A_{11}}{h}$ is the unknown:

$$\bar{A}_{11}{}^2 \frac{(k-1)^2}{4k} - \bar{A}_{11}[(k+1)(U_1 + U_4) - E_x] + k(U_1 + U_4)^2 = 0, \tag{4.13}$$

The roots of the equation express the term $\frac{A_{11}}{h}$ as a function of E_x and k.

Thus, taking E_x and k as parameters, we can calculate $\bar{A}_{11}$ and further with help of (4.11) the entire normalized membrane stiffness sub-matrix $\bar{A}$. When it is known the in-plane engineering constants of the panel (μ_{xy}, μ_{yx} and G_{xy}) can be calculated as the following:

$$\mu_{xy} = \frac{\bar{A}_{12}}{\bar{A}_{22}}, \mu_{yx} = \frac{\bar{A}_{12}}{\bar{A}_{11}}, G_{yx} = \bar{A}_{66}. \tag{4.14}$$

In order to calculate the stiffnesses of the panel for the equation (4.2) at the upper level we need to define its total thickness h. Also knowing the thickness, it is possible to calculate the extensional stiffness sub-matrix according to (4.11). Bending stiffness sub-matrix can be calculated via (4.1).

Finally, at the upper level only 3 parameters (E_x, k and h) are required to fully define the stiffness of a laminate, which has constant thickness and stacking sequence. It is very clear and convenient from the engineering point of view. These parameters are independent opposed to the lamination parameters in (1.1) which are not independent. Moreover, the stacking sequence of the laminate is not important at this level. Also, it is very simple to define the limits for these parameters.

Applying this approach to the global FE model only the lamina properties and the total thickness of the laminate are required to define each composite sub-structure.

In order to fully define a complex structure, which has variable stiffness, $3 \times N_{sub}$ parameters are required, where N_{sub} is the number of sub-structures or structural elements with different stiffness matrices.

The thickness h of each sub-structure is a discreet parameter, but E_x and k are the continuous ones. In general this is the reason that the upper-level optimization method should be able to deal with mixed integer-continuous parameters, however, as it was shown above (see (4.5)) the laminate total thickness is not so important at the upper level since the product hE_x is a continuous quantity. Therefore, h can be set as continuous at the upper level and well-known classical multi-parametric optimization methods can be used here.

4.1.2 Limits for E_x and k parameters

Evidently, the value of elastic modulus E_x of a laminate could not be whatever when particular lamina properties are defined. It has some maximum and minimum possible limits, which depend on the stacking sequence.

Similarly, the limits of k parameter must be defined since the orthogonal elastic modulus E_y defined via (4.10) should be in correlation with E_x modulus.

Finally, the stiffness sub-matrix A should be positive definite and all its components should be strictly positive also. However, some combinations of E_x and k parameters could define sub-matrix A, which is not positive definite. This case will lead to fatal error during the global FE model analysis.

Thus, it is very important to define the limits for these parameters.

For E_x it is quite simple. It depends on lamina material. There are two cases:

1) The lamina material is a balanced fabric, and its longitudinal and transversal properties are equal:
   a) the maximum of E_x is equal to the longitudinal elastic modulus E_1 of the lamina,
   b) the minimum is equal to the E_x elastic modulus of a cross-ply $[\pm 45]_{ns}$ laminate,
2) The lamina material is an unbalanced fabric or a unidirectional material:
   a) the maximum of E_x is also equal to the longitudinal elastic modulus E_1 of the lamina,
   b) the minimum is equal to the transversal elastic modulus E_2 of the lamina.

It worth mentioning that in the first case the expressions $E_x = E_y$ and $k = 1$ are true for any symmetric stacking sequence.

To define the limits for the k parameter let's get back to the definition (4.9) of the sub-matrix A. Since the nonzero components A_{ij} depend on $\bar{\xi}_1$ and $\bar{\xi}_2$ parameters and U_i constants, it is better to define the limits for the $\bar{\xi}_1$ and $\bar{\xi}_2$ parameters first. These parameters can be calculated as follows:

$$\begin{aligned} \bar{\xi}_1 &= \frac{\xi_1}{h} = \sum_{i=1}^{n} \frac{h_i}{h} \cos 4\theta_i, \\ \bar{\xi}_3 &= \frac{\xi_3}{h} = \sum_{i=1}^{n} \frac{h_i}{h} \cos 2\theta_i, \end{aligned} \tag{4.15}$$

where h_i and θ_i - thickness and orientation angle of co-oriented batch of plies.

It is always true that $\sum_{i=1}^{n} \frac{h_i}{h} = 1$ and $-1 \le \cos 2\theta_i \; (or \; \cos 4\theta_i) \le 1$. Therefore, the next cases can be written:

1) all plies have orientation $\theta_i = 0°$, then $\cos 2\theta_i = 1$ and $\cos 4\theta_i = 1$: $\bar{\xi}_1 = \bar{\xi}_3 = 1$,

2) all plies have orientation $\theta_i = \pm 22{,}5°$, then $\cos 2\theta_i = \sqrt{2}/2$ and $\cos 4\theta_i = 0$: $\bar{\xi}_1 = 0, \bar{\xi}_3 = \sqrt{2}/2$,
3) all plies have orientation $\theta_i = \pm 45°$, then $\cos 2\theta_i = 0$ and $\cos 4\theta_i = -1$: $\bar{\xi}_1 = -1, \bar{\xi}_3 = 0$,
4) all plies have orientation $\theta_i = 90°$, then $\cos 2\theta_i = -1$ and $\cos 4\theta_i = 1$: $\bar{\xi}_1 = 1, \bar{\xi}_3 = -1$.

Before defining the limits for k parameter it is good to write the next equations:

$$\begin{aligned} U_1 + U_2 + U_3 &= Q_{11}, \\ U_1 - U_2 + U_3 &= Q_{22}, \end{aligned} \tag{4.16}$$

Then for the general case, when the lamina material has different longitudinal and transversal properties, the limits for k parameter can be calculated as the following:

1. $\theta_i = 90°, \bar{\xi}_1 = \bar{\xi}_3 = 1$: according to (4.9) $A_{11} = h(U_1 + U_2 + U_3)$, $A_{22} = h(U_1 - U_2 + U_3)$, then taking into account (4.16) one can write

$$k = \frac{E_x}{E_y} = \frac{A_{11}}{A_{22}} = \frac{U_1 + U_2 + U_3}{U_1 - U_2 + U_3} = \frac{Q_{11}}{Q_{22}} = \frac{E_1}{E_2}.$$

2. $\theta_i = 0°, \bar{\xi}_1 = 1, \bar{\xi}_3 = -1$: similarly to the first case

$$k = \frac{U_1 - U_2 + U_3}{U_1 + U_2 + U_3} = \frac{Q_{22}}{Q_{11}} = \frac{E_2}{E_1}.$$

The other cases give values falling between these two. Finally, it can be written, that

$$k \epsilon \left[\frac{E_2}{E_1}; \frac{E_1}{E_2}\right]. \tag{4.17}$$

4.2 GA level

At the lower level the structure is divided into several sub-structures, where the sub-structures are optimized in detail. However, the optimization in different sub-structures can be done in parallel, the integrity of the entire structure should be kept. From the manufacturing point of view, in general, this means at least the neighboring sub-structures should have conforming stacking sequences what is called blending.

According to Section 1 the stochastic optimization methods have very high ranking in application to composite structures. The GAs are between them. They have vital advantages such as the ability to manage large number of design variables and to find the global optimum, they do not require gradient information, have low cost in parallel optimization and are very simple for realization. That why a GA was chosen for the optimization of the laminates stacking sequances in this work.

There exist two basic approaches to obtain blended designs in the neighboring sub-structures using GAs in parallel realization [27]. The general idea of the first one is in generating

of multiple populations, which correspond to stacking sequences of different sub-structures (e.g., panels) and are optimized in parallel. The blended designs are obtained with help of evolutionary pressures acting on each population from neighboring ones. The pressure is realized by different methods, e.g., by random migration of individuals between the neighboring populations [40] or addition of continuity constraints [41]. However, the main disadvantage of this approach is in that it does not guarantee fully blended designs corresponding to the global optimum.

The second approach is based on so called "guides". Within this approach each guide corresponds to a stacking sequence, which has the maximum possible thickness for entire structure. The stacking sequence for each sub-structure is obtained from the guide by deleting the redundant layers until the strength/stiffness/etc. criteria are violated. At the beginning the GA generates a population of guides, which are exposed to genetic operations then. At the end the global optimum design is represented by the best guide combined with an array, which contains the integer numbers corresponding to number of plies within laminates (or to number of plies subtracted from the guide) of each sub-structure. The main advantage of this approach is in that all designs considered are always blended right from the beginning of the optimization algorithm. In such a way the dimensionality of the problem becomes much less, and the continuity constraints are not required anymore. This fact simplifies the problem solution with help of GAs or alternative methods. However, this simplification is obtained at the expanse of flexibility loss when trading the degree of blending against weight. The reliability and resolution of this approach for sure in some extent covers above noted loss, since the slightly unblended designs should not be much lighter than the perfectly blended ones.

Because of the spoken advantages the guide-based approach was chosen for optimization of stacking sequences of laminates at the GA level in the present study.

The guides are represented by stacking sequences in form of one-dimensional arrays with fixed length where each element is the orientation angle of a corresponding ply. Since only orthotropic laminates are considered in this paper the guides represent only a half of a stacking sequence (see Fig. 4.1). The angles can be chosen by a user depending on the manufacturing requirements. Each guide is accompanied with a fitness value which helps the guides from the same generation to compete.

The first generation of guides is created randomly taking into account manufacturing and design constraints such as for balance, orthotropy, etc. [37].

The key operators for any GA are the selection, crossover, mutation, and the replacement ones [39]. Also, it is very important how the fitness value is calculated. At the lower level it is calculated using the upper-level objective function (4.2). However, this function corresponds to a particular cross-section only but not to the entire structure. The integral function at the GA level for the entire structure can be calculated as follows:

$$F = \sqrt{\sum_{k=1}^{N_s} f_k}, \tag{4.18}$$

where f_k – value of the function (4.2) calculated for a $k-th$ cross section, N_s – total number of defined cross sections of the structure.

Function (4.18) shows the difference in stiffness between upper and lower levels. However, it does not consider the weight of the structure. Therefore, the objective function at the upper level can be as follows:

$$F_{obj} = (1 + F)W, \tag{4.19}$$

The selection operator is based on the modified elitist strategy. The first of the parent individuals is chosen from the population randomly. The second one is chosen from a group of the best individuals of the population (the number of individuals in the group is chosen during the customization of the algorithm) in such a way that the Euclidian distance (4.20) between it and the first parent is the smallest within this group. Such a strategy prevents premature convergence.

$$d = \sqrt{\sum_{j=1}^{n} \left(\theta_{1j} - \theta_{2j}\right)^2}, \tag{4.20}$$

where n – total number of plies in a guide, θ_{1j} and θ_{2j} – orientation angles of corresponding $j-th$ plies of the first and the second parents.

The crossover operator happens with a high probability $P_c \geq 0.95$, which can be defined by a user. In this work two-point crossover is used (see Fig. 4.4). The positions "pos 1" and "pos 2" are defined randomly, while always "pos 1" ≠ "pos 2". Such an operator is used to have a higher probability to influence the thinnest sub-structures.

The mutation operator is applied to each of the children right after the crossover. This operator replaces the orientation angle of a randomly chosen ply with a new angle, which is randomly chosen from the group of angles allowed by the user. For sure the balanced stacking sequence is kept. It happens with a small probability $P_m \leq 0.1$ which can be adjusted by a user.

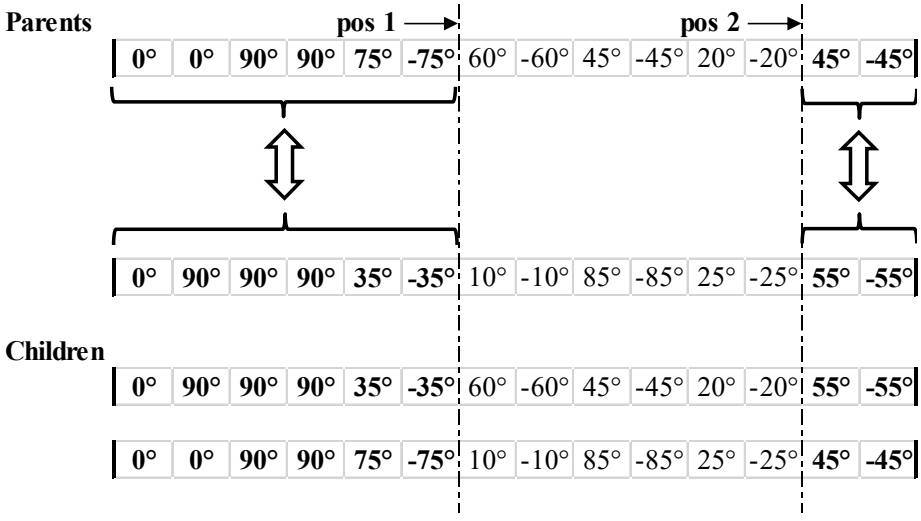

Fig. 4.4 Crossover operation

After mutation has been done, the children individuals are sent to one-dimensional optimization of the thickness of each sub-structure. As the result of this optimization the fitness values (4.19) of the new guides are calculated.

The replacement operator does the next actions:

- randomly choose C_f groups (each group has S individuals) of individuals from the population as candidates to be replaced by the new ones (children),
- within each group a candidate to be replaced is chosen, which has the smallest Euclidian distance (4.20) to the new individual, thus C_f candidates are found,
- the new individual replaces one of the found C_f candidates, which has the worst fitness value.

The above-described operators are repeated until a stop criterion is reached. Three stop criteria are used in this thesis. The simplest one is met when GA has reached a predefined number of generations. It, however, requires experiments and gaining experience in defining such a number, which can guarantee the global optimum to be found.

The second stop criterion is met when the predefined number of best guides (N_{best}) within the actual population have difference between their fitness and the average fitness not more than the predefined value ν, see (4.21). The ν value is defined by the user.

$$\begin{gathered} \frac{F_{obj\,i} - \bar{F}_{obj}}{\bar{F}_{obj}} \cdot 100\% \le \nu, \\ i = 1 \dots N_{best}\,, N_{best} < N_{tot}, \\ \bar{F}_{obj} = \frac{1}{N_{best}} \sum_{i=1}^{N_{best}} F_{obj\,i}, \end{gathered} \tag{4.21}$$

where $F_{obj\,i}$ – fitness value (4.19) of the *i-th* best guide, N_{tot} – the total number of guides in population.

The third stop criterion is met when the best guide has its fitness less than the predefined value $F_{obj\,max}$. The value of $F_{obj\,max}$ is defined by the user.

The described criteria can be applied apart or as a combination. However, the first criterion is always applied to limit the maximum number of iterations.

4.3 Lower-level one-dimensional parallel optimization

At the lower level within each sub-structure, which has constant thickness, the stacking sequence is already predefined by the guide. The only parameter, which can be varied, is the number of plies or thickness of the sub-structure. While the number of plies is being changed, the elastic characteristics of the sub-structure is being changed also. As it was shown in Subsection 4.1 when the geometry of the sub-structure is fixed its stiffnesses is defined by the (hE_x) and (hG_{xy}) parameters only. Therefore, it is not necessary calculating the stiffnesses as in (4.2). It is simpler to use (hE_x) and (hG_{xy}) values in the calculation of lower-level objective

function when each sub-structure is optimized independently. Thus, the objective function (4.2) can be modified as follows:

$$f_d = \left[\frac{(hE_x) - (hE_x)^*}{(hE_x)^*}\right]^2 + \left[\frac{\left(hG_{xy}\right) - \left(hG_{xy}\right)^*}{\left(hG_{xy}\right)^*}\right]^2, \tag{4.22}$$

where the star symbol (*) still denotes the parameter's value at the upper level.

The one-dimensional minimization of the function f_d, which is done in parallel for each sub-structure, is the objective of the lower-level optimization. The optimized parameter is the thickness/number of plies of the sub-structure. The constraints, which usually applied at this level, are the strength and the buckling of the sub-structure. The Gold Section method adopted to an integer variable (number of plies) is used here.

The calculation of structural responses (stress, buckling factors, etc.) is done by FE analysis of the local models in parallel.

4.4 Coordination between upper and lower levels

Since the distribution of loads depends on the stiffnesses distribution, the difference in stiffness (4.2) means the internal forces distributions at the upper and lower levels are not correlated and the entire optimization lacks convergence.

The authors in [2] propose a coordination procedure, which applies the next upper-level coordination constraints (one for each lower-level optimization):

$$g_k = f_k^U - (1-\varepsilon) f_{ok}^L \leq 0, \tag{4.23}$$

where f_{ok}^L - the most recent value of the lower-level objective function (i.e., the optimum value of eq. (4.2) for the $k-th$ cross-section, f_k^U - estimate of the change in f_{ok}^L that would be caused by a change in the upper-level design variable values, and ε - specified tolerance defined as the coordination parameter.

Calculation of f^U requires quite expensive sensitivity analysis, which in case of complex composite structures becomes very difficult also.

In the present work the coordination is achieved through the minimization of the objective functions at the lower (4.22) and the GA levels (4.18). This minimization means that the difference in stiffness at the upper and lower levels aims to a minimum.

Of course, in some cases/iterations it is not possible to achieve good correlation between the upper and lower levels. However, it does not always mean a lower-level design violates the upper-level constrains. To check if a lower-level design does not violate the upper-level constrains another global model is introduced into the algorithm. This model is not the same as for defining stiffness distribution which was described above. It takes into account the optimal desings of all sub-structures found at the lower level within for the particular upper-level optimization step. The FEA of this model is performed only for the best guides and thickness distribution found at the GA level. If the upper-level constrains are not violated these designs could be considered as the potential optima.

5 Application of the developed optimization methodology

To parallelize the lower-level calculations and simplify the interconnection between the coded optimization algorithm and FEA software a commercial software Noesis Optimus by Noesis Solutions was used. This software allows to integrate many different engineering software and domestic codes into one powerful optimization algorithm. It is also possible to integrate an own newly developed domestic optimization algorithm into Optimus. Optimus allows building a flexible multilevel optimization algorithm, which can be adopted for optimization of different complex structures.

To calculate responses of a structure MSC Nastran is used. The optimization methods for GA and lower levels are programmed using Python 2.7 language. The optimization method at the upper level is chosen from those proposed by Noesis Optimus software, which are built-in in it.

To validate the developed optimization methodology, it was applied to a simple optimization problem, which was proposed in [23]. Thus, the optimization results can be compared to those presented in this publication.

A simple wing-box structure is used for optimization in this problem. The wing-box is unswept, untapered and rectangular with dimensions 3543.3×2240.3×381 mm (Fig. 5.1).

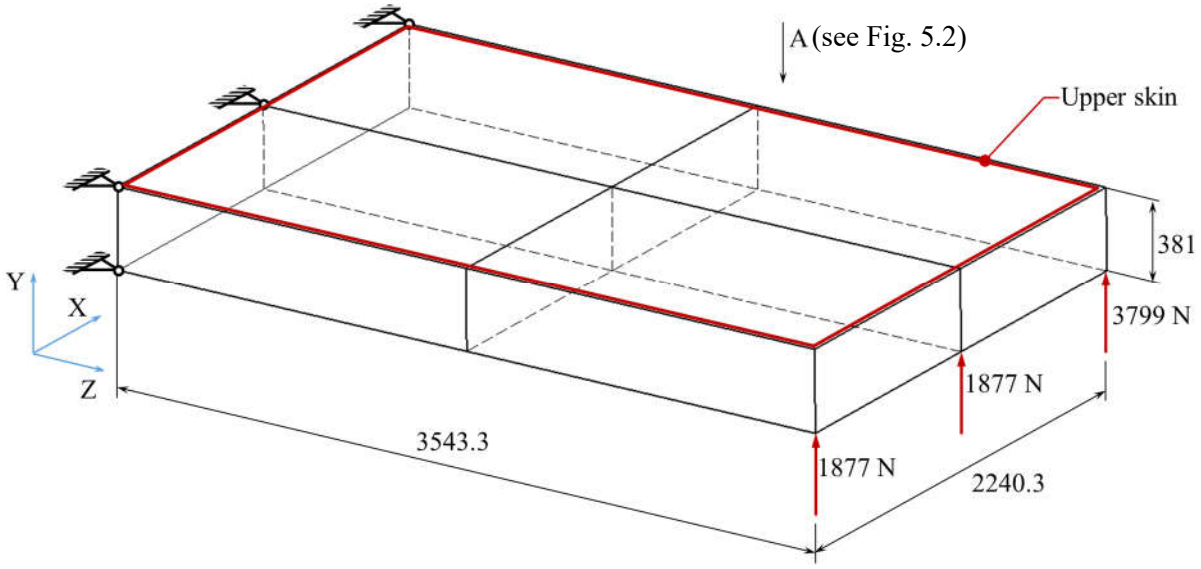

Fig. 5.1 Simple wing structure model

The spars and the ribs divide the top and bottom skin of the wing box into four panels of equal size. Only the top skin panels are being optimized, all other parts are fixed to the design $[\pm 45_5/45]_s$. The upper panels' arrangement is shown in Fig. 5.2. In the global FE model all structural elements are modeled using shell elements. All nodes of the wing root have all their translational DOFs fixed. The upward lift force is modeled by three concentrated loads of magnitudes 3799 N, 1877 N and 1877 N applied at three lower nodes (left, middle and right) at the free wing tip. The lamina material is the graphite/epoxy T300/5208 (see Tab. 5.1). The upper skin design only is subjected to strength and buckling constraints.

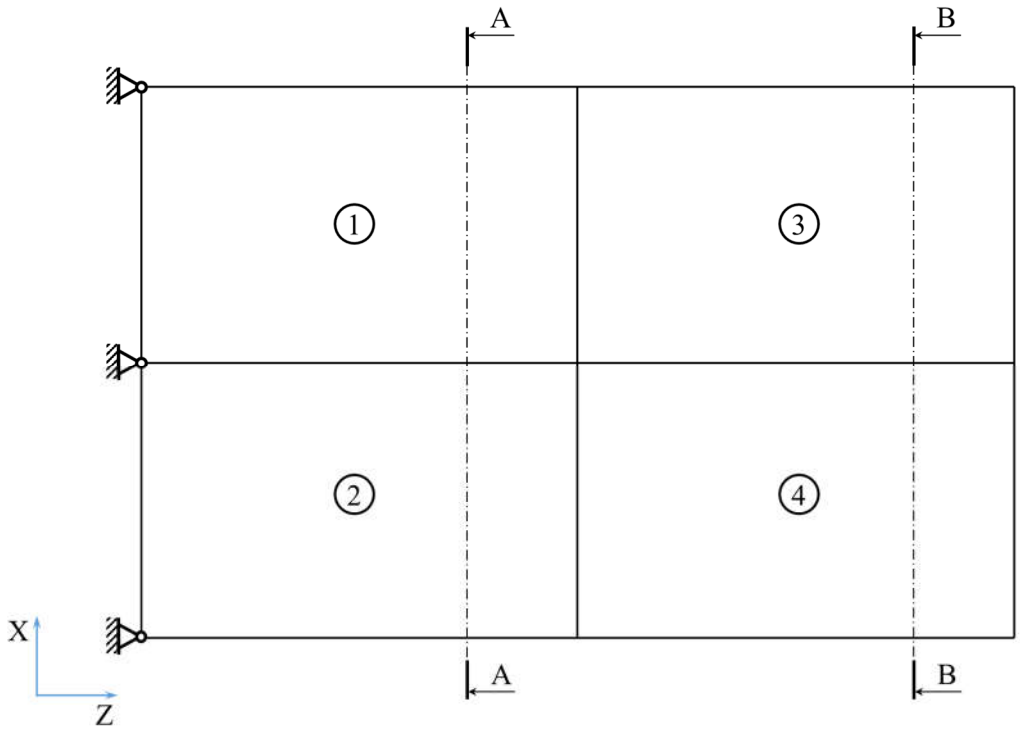

Fig. 5.2 Panels' layout for the upper skin (view A, see Fig. 5.1)

In general, the complexity of FE models at both levels is chosen by a designer according to the traditions and regulations/standards of a particular company. At the upper level it should be a rough model, where the skin and webs should be modeled with shell elements and the stiffeners with beam or bar elements. The mesh size should be rough also. At the lower level the FE model should be detailed enough to obtain structural responses with a precision required for assessing strength, buckling, etc., constraints of all important structural elements.

The descriptions of the upper-level and lower-level FE models in detail are provided below in the subsequent subsections.

Tab. 5.1 Lamina material properties of graphite/epoxy T300/5208

Property	**Unit**	**Value**
Longitudinal elastic modulus, E_1	Pa	128×10^9
Transversal elastic modulus, E_2		13×10^9
In-plane shear modulus, G_{12}		6.4×10^9
Poisson ratio, μ_{12}	-	0.3
Longitudinal allowable strain, ε_{1a}		0.008
Transversal allowable strain, ε_{2a}		0.029
In-plane shear allowable strain, γ_{12a}		0.015
Lamina thickness, h_0	m	0.127×10^{-3}
Density, ρ	kg/m^3	1402.48

5.1 Upper-level construction

5.1.1 Optimization method

The upper-level optimization method is chosen from those available in the Optimus software. It is Efficient Global Optimization method (EGO) [42]. EGO "is a hybrid optimization algorithm in which an interpolating response surface model is built in every iteration and new simulation points are added based on the result of an optimization that is performed on the response surface model.

Since the Noesis Optimus software is used in this work, its terminology will be used further. Each optimization step where a new structural design is calculated is called experiment.

The response surface model that is chosen for the EGO algorithm is the Kriging model. The advantages of this type of model are the fact that the model will interpolate through all experiments, and the fact that an estimate can be made of the prediction error of the model. This error will be zero in the interpolation points, small in points that are close to an interpolation point, and bigger if the distance to any interpolation point becomes larger. This property of the Kriging model is used in the decision where to add a point in the next iteration.

A new simulation point will be added in a promising region in the design space, i.e.:

- in regions where the objective is expected to become lower because the response surface predicts a lower value,
- or in regions where the objective gets a higher probability to become lower because the accuracy of the response surface is low due to the fact that in this region points are far from the known points,
- of course, no new points will be added in regions where the probability is high that at least one of the constraints is violated."

The method has options shown in *Fig. 5.3* [42]. The values chosen for the example problem optimization are shown in the figure. The meanings of the basic control parameters are the next:

- Number experiments for first LH - this is the number of experiments in the first space-filling Latin Hypercube that has to be performed,
- Random seed - a 0 value will initialize the random generator for the algorithm based on the time of the system. A positive value will initialize it with this specified value,
- Maximum number of evaluations - the maximum number of function evaluations allowed for this optimization,
- Tolerance - termination criterion. The algorithm will stop if one of the previous experiments p_j is found for which

$$\frac{p^j - p^j{}_k}{p^j{}_k} < \delta \tag{5.1}$$

for all dimensions k, with p the newly proposed point. So, EGO stops if it tries to add a new point that is too close to one of the previous experiments.

- Number of Sigma. A low value means the algorithm will more behave as a local optimizer. A higher value (e.g., 3) means the algorithm behaves more like a global optimizer.

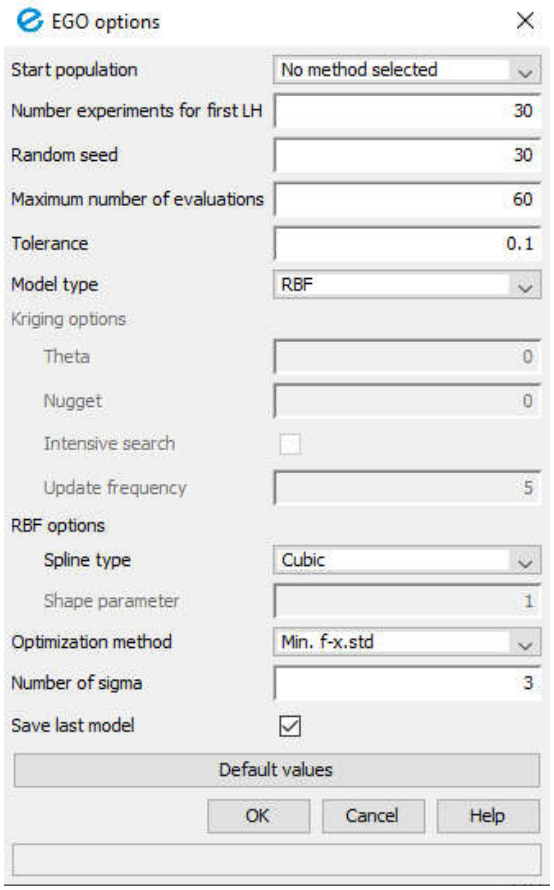

Fig. 5.3 EGO Options

The weight of the structure is minimized at this level.

5.1.2 Cross-section geometry of the wing box

To calculate the fitness values of guides (4.18) the stiffnesses of the structure in the important cross-sections should be calculated. The wing box structure (Fig. 5.1) has two such cross-sections. The first one is A-A, through the panels 1 and 2 and the second one is B-B, through the panels 3 and 4 (see Fig. 5.2). The geometry of the cross-sections is shown in *Fig. 5.4*. It is the same, except the thicknesses of the optimized panels. The dimensions a =1120.15 mm and c = 381 mm according to Fig. 5.1. The thickness of the spars and lower skin (t = 2.794 mm) is constant and corresponds to $[\pm 45_5/45]_s$ stacking sequence. The thickness $t_{1(3)}$ corresponds to the panel 1 in the A-A cross-section and to the panel 3 in the B-B cross-section. Similarly, the thickness $t_{2(4)}$ corresponds to the panels 2 and 4. The longitudinal elastic and in-plane shear moduli of these structural elements have similar indices: $E_z, G_{zx} (or\ G_{yz})$ – for spars and lower skin, $E_{z1(3)}, G_{zx1(3)}$ - for panels 1 and 3, $E_{z2(4)}, G_{zx2(4)}$ – for panels 2 and 4.

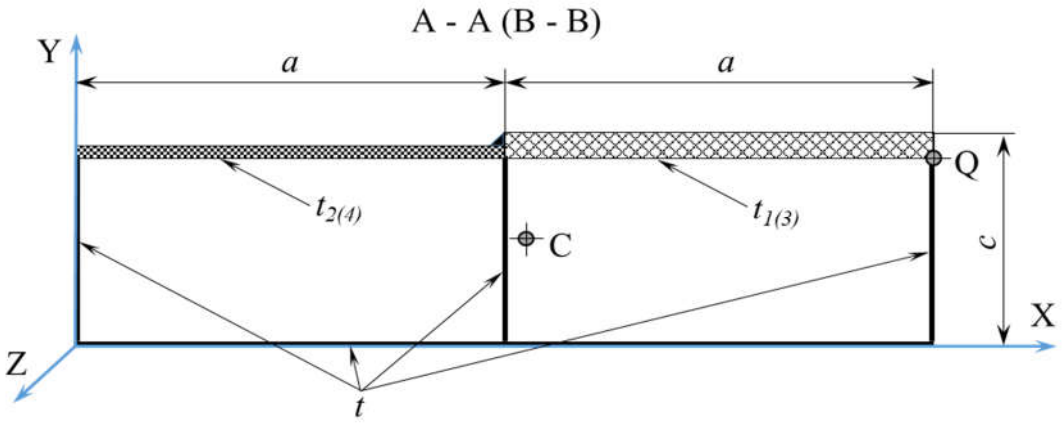

Fig. 5.4 Cross-section of the wing box structure

The extensional (longitudinal) stiffness of the cross-sections can be calculated as the following:

$$AE_{z1(2)} = \oint tE_z ds = tE_z(3c + 2a) + a\left(t_{2(4)}E_{z2(4)} + t_{1(3)}E_{z1(3)}\right). \tag{5.2}$$

The first moments of area are the following:

$$\begin{aligned} EI_{x1(2)} &= \oint tE_z y ds = \frac{3}{2}tE_z c^2 + ca\left(t_{2(4)}E_{z2(4)} + t_{1(3)}E_{z1(3)}\right), \\ EI_{y1(2)} &= \oint tE_z x ds = 3tE_z ac + 2tE_z a^2 + \frac{a^2}{2}t_{2(4)}E_{z2(4)} + \frac{3}{2}a^2 t_{1(3)}E_{z1(3)}. \end{aligned} \tag{5.3}$$

Then the centroid C coordinates can be simply calculated:

$$x_{c1(2)} = \frac{EI_{y1(2)}}{AE_{z1(2)}}, y_{c1(2)} = \frac{EI_{x1(2)}}{AE_{z1(2)}}. \tag{5.4}$$

The bending stiffnesses can be calculated as follows, starting from the point Q:

$$\begin{aligned} EJ_{x1(2)} &= \oint tE_z y^2 ds = tE_z c^3 + c^2 a\left(t_{2(4)}E_{z2(4)} + t_{1(3)}E_{z1(3)}\right), \\ EJ_{y1(2)} &= \oint tE_z x^2 ds = tE_z\left(5a^2 c + \frac{8}{3}a^3\right) + \frac{a^3}{3}\left(t_{2(4)}E_{z2(4)} + 7t_{1(3)}E_{z1(3)}\right). \end{aligned} \tag{5.5}$$

The bending stiffnesses in relation to the centroid C are the next:

$$\begin{aligned} EJ_{x01(2)} &= EJ_{x1(2)} - {y_{c1(2)}}^2 AE_{z1(2)}, \\ EJ_{y01(2)} &= EJ_{y1(2)} - {x_{c1(2)}}^2 AE_{z1(2)}. \end{aligned} \tag{5.6}$$

There does not exist a simple formula for torsional stiffness of a multi-closed cross-section, since it is a statically indeterminate. If we apply a unit torsional moment to the cross section, the torsional stiffness can be defined as the following:

$$GJ_z = \frac{M_t}{\varphi}, \tag{5.7}$$

where $M_t = 1Nm$, φ – specific angle of torsion of the A-A (B-B) cross-section, rad/m.

In order to define the angle φ the method of redundant reactions should be applied. The cross-section is cut as shown in Fig. 5.5. The redundant reaction moments M_I and M_{II} appear in each opened sub-part of the cross-section.

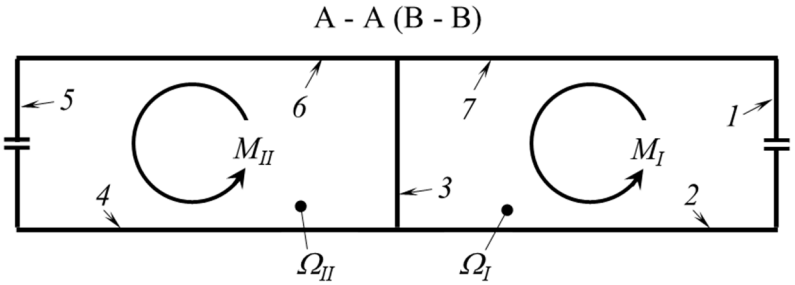

Fig. 5.5 Redundant reactions

According to the method of redundant reactions two equations with unknown moments M_I and M_{II} can be written:

$$\begin{aligned} \delta_{11}M_I + \delta_{12}M_{II} + \Delta_{1p} &= \varphi_I, \\ \delta_{21}M_I + \delta_{22}M_{II} + \Delta_{2p} &= \varphi_{II}, \end{aligned} \tag{5.8}$$

where

$$\begin{aligned} \delta_{11} &= \frac{1}{4{\Omega_I}^2}\sum_{i=1}^{n_1}\frac{\Delta l_i}{(Gt)_i}, \\ \delta_{22} &= \frac{1}{4{\Omega_{II}}^2}\sum_{i=1}^{n_2}\frac{\Delta l_i}{(Gt)_i}, \\ \delta_{12} = \delta_{21} &= \sum_{i=1}^{n}\frac{1}{2\Omega_I}\frac{1}{2\Omega_{II}}\frac{\Delta l_i}{(Gt)_i}, \\ \Delta_{1p} = \Delta_{2p} &= 0, \\ \varphi_I = \varphi_{II} &= \varphi, \end{aligned} \tag{5.9}$$

Ω_I, Ω_{II} – internal area swept by the first and the second sub-parts of the cross-section correspondingly, Δl_i – length of the $i-th$ portion of the cross-section (see numbers 1-7 in Fig. 5.5), $(Gt)_i$ – in-plane shear modulus multiplied by the thickness of the $i-th$ portion of the cross-section, n_1, n_2 – number of the portions the $I-st$ and the $II-nd$ sub-parts of the cross-section correspondingly divivded into, n – total number of portions the cross-section divided into.

From the other hand the sum of M_I and M_{II} moments should be equal to the applied moment, i.e.

$$M_I + M_{II} = M_t = 1. \tag{5.10}$$

The eq. (5.8) and (5.10) together give the next system of equations:

$$\begin{cases} \delta_{11}M_I + \delta_{12}M_{II} = \varphi, \\ \delta_{21}M_I + \delta_{22}M_{II} = \varphi, \\ M_I + M_{II} = 1. \end{cases} \tag{5.11}$$

Solving the system one can get:

$$\begin{aligned} M_I &= \frac{\delta_{12} - \delta_{22}}{2\delta_{12} - \delta_{11} - \delta_{22}}, \\ M_{II} &= \frac{\delta_{12} - \delta_{11}}{2\delta_{12} - \delta_{11} - \delta_{22}}. \end{aligned} \tag{5.12}$$

Then the the angle of torsion of the cross-section will be the next:

$$\varphi = \frac{\delta_{11}(\delta_{12} - \delta_{22}) + \delta_{12}(\delta_{12} - \delta_{11})}{2\delta_{12} - \delta_{11} - \delta_{22}}. \tag{5.13}$$

According to (5.7) the torsion stiffness of the cross-section can be calculated as the following:

$$GJ_z = \frac{M_t}{\varphi} = \frac{1}{\varphi} = \frac{2\delta_{12} - \delta_{11} - \delta_{22}}{\delta_{11}(\delta_{12} - \delta_{22}) + \delta_{12}(\delta_{12} - \delta_{11})}. \tag{5.14}$$

For calculation of the coeffitients δ_{ij} Tab. 5.2 is drawn up.

Taking into account that $\Omega_I = \Omega_{II} = ac$ and using Tab. 5.2 the coefficients (5.9) become the next

$$\begin{aligned} \delta_{11} &= \frac{1}{4a^2c^2}\left(\frac{2c + a}{tG_{zx(yz)}} + \frac{a}{t_{1(3)}G_{zx1(3)}}\right), \\ \delta_{22} &= \frac{1}{4a^2c^2}\left(\frac{2c + a}{tG_{zx(yz)}} + \frac{a}{t_{2(4)}G_{zx2(4)}}\right), \\ \delta_{12} &= \delta_{21} = -\frac{1}{4a^2c^2}\frac{c}{tG_{zx(yz)}}. \end{aligned} \tag{5.15}$$

After substitution (5.15) into (5.14) and mathematical transformations the formula for torsional stiffness gets its final form

$$GJ_{z1(2)} = \frac{4a^2c^2\left(\frac{6c + 2a}{tG_{zx(yz)}} + \frac{a}{t_{1(3)}G_{zx1(3)}} + \frac{a}{t_{2(4)}G_{zx2(4)}}\right)}{\left(\frac{2c + a}{tG_{zx(yz)}} + \frac{a}{t_{1(3)}G_{zx1(3)}}\right)\left(\frac{2c + a}{tG_{zx(yz)}} + \frac{a}{t_{2(4)}G_{zx2(4)}}\right) - \left(\frac{c}{tG_{zx(yz)}}\right)^2}. \tag{5.16}$$

Tab. 5.2 For calculation of δ_{ij} coeffitients

Parameter (*Fig. 5.4*)	Parameter value at $i-th$ portion						
	Portion number (Fig. 5.5)						
	1	2	3	4	5	6	7
t_i	t	t	t	t	t	$t_{2(4)}$	$t_{1(3)}$
G_i	$G_{zx(yz)}$	$G_{zx(yz)}$	$G_{zx(yz)}$	$G_{zx(yz)}$	$G_{zx(yz)}$	$G_{zx2(4)}$	$G_{zx1(3)}$
Δl_i	c	a	c	a	c	a	a
$\frac{\Delta l_i}{(Gt)_i}$	$\frac{c}{tG_{zx(yz)}}$	$\frac{a}{tG_{zx(yz)}}$	$\frac{c}{tG_{zx(yz)}}$	$\frac{a}{tG_{zx(yz)}}$	$\frac{c}{tG_{zx(yz)}}$	$\frac{a}{tG_{zx2(4)}}$	$\frac{a}{tG_{zx1(3)}}$

5.1.3 Description of the global FE model

Before building the upper-level optimization workflow the initial global FE model should be created. It could be a rough simplified model (Fig. 5.6).

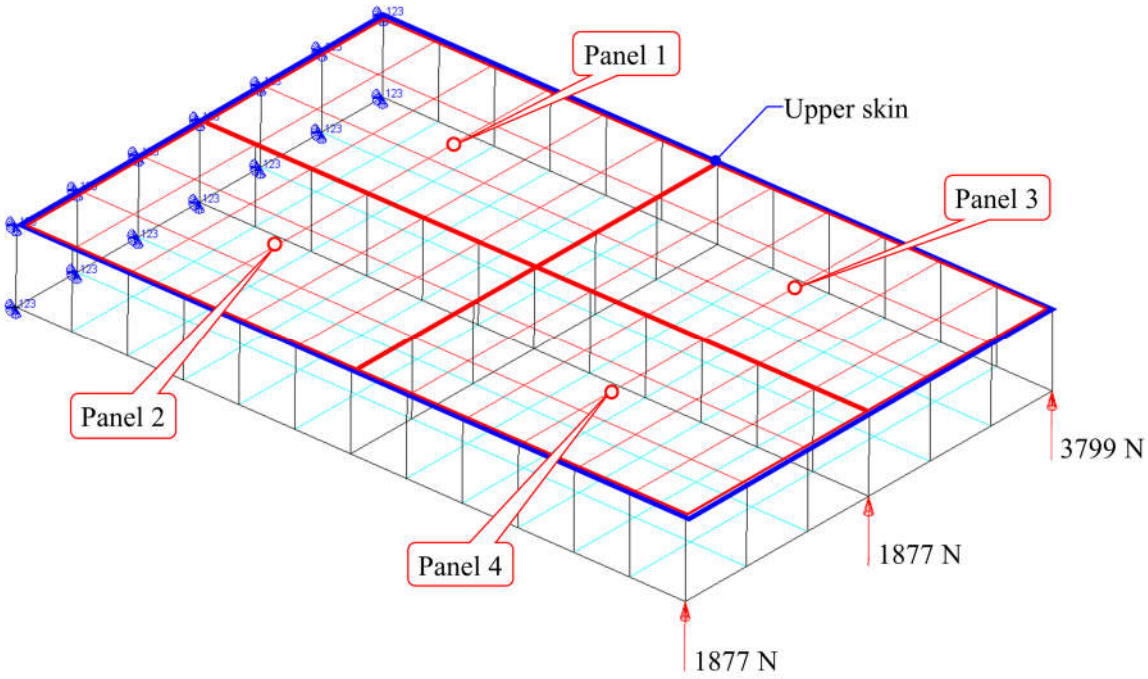

Fig. 5.6 Global FE model

In case of the example problem described above the wing structure was modeled using MSC Patran pre-processor. The structural elements, which are not being optimized, have their stacking sequences fixed to the $[\pm 45_5/45]_S$ design. Therefore, this laminates are modeled using PCOMP card (see Appendix A). The lamina material used in PCOMP cards is modeled using MAT8 card (see Appendix A). The upper skin panels (to be optimized) are modeled using shell elements with an equivalent section's property PSHELL (see Appendix A). The equivalent section's property is defined using two material fields (Membrane Material and Bending Material) and thickness. The other fields are left blank. These material fields are set to the same MAT2 material card (see Appendix A). In such a way the smeared stiffness approach (see Subsection 4.1 and expression (4.1)) is modeled. The MAT2 card allows defining a composite

material through its extensional stiffness sub-matrix A [43]. It is used for defining the composite materials of each upper skin panel. All nodes of the wing root have all their translational DOFs fixed. Three concentrated loads of magnitudes 3799 N, 1877 N and 1877 N are applied at three lower nodes (left, middle and right) at the free wing tip. The average mesh size is 350 mm.

The listing of the BDF file for the initial global FE model is shown in Appendix A.

5.1.4 Description of the upper-level analysis

Within the workflow of the upper-level analysis the following steps are performed (see Fig. 5.7):

1. The input data for the optimized structure (lamina material properties, geometry, loading, etc.) are read.
2. The lamina stiffness matrix Q is calculated.
3. The material constants U_i are calculated according to (4.7).
4. For each optimized upper panel do:
   a. Solve quadratic equation according to (4.13) using constants U_i and parameters E_{xi}, k_i, h_i defined by the upper-level optimization.
   b. The nonzero components of extensional stiffness sub-matrix A_{ij} are calculated according to (4.11).
   c. If the sub-matrix A is positive definite and its components are positive the Python script "MAT2_writer.py" (see Appendix B) generates BDF file, which contains MAT2 material card for the panel. This file is inserted to the main BDF file of the global model using INCLUDE statement. Simultaneously to that geometric characteristics of the wing box cross-sections A-A and B-B are calculated according to (5.2) - (5.16) in parallel.
   d. Otherwise, the experiment is taken as failed and is not taken into account in further steps. In this case the analysis is finished, and the focus is returned to the upper-level optimization.
5. The loads and thickness of the panel are being written into the main BDF file of the global model.
6. Static stress analysis of the global model using MSC Nastran (SOL101) is performed.
7. The required results (translational and rotational displacements of nodes at the edges of the upper panels are extracted by Optimus from the result F06 file.
8. If fatal errors are found in result F06 file the experiment is taken as failed and is not taken into account in further steps. In this case the analysis is finished, and the focus is returned to the upper-level optimization.
9. Otherwise, the extracted displacements are transferred to the GA level and then to the lower level, where they are used as boundary conditions in the local models. The EJ_z, EJ_y, GJ_z global stiffnesses, E_{xi} moduli and h_i thicknesses are transferred to the GA level and then to the lower level also.
10. The GA level is started (see Subsection 5.2).

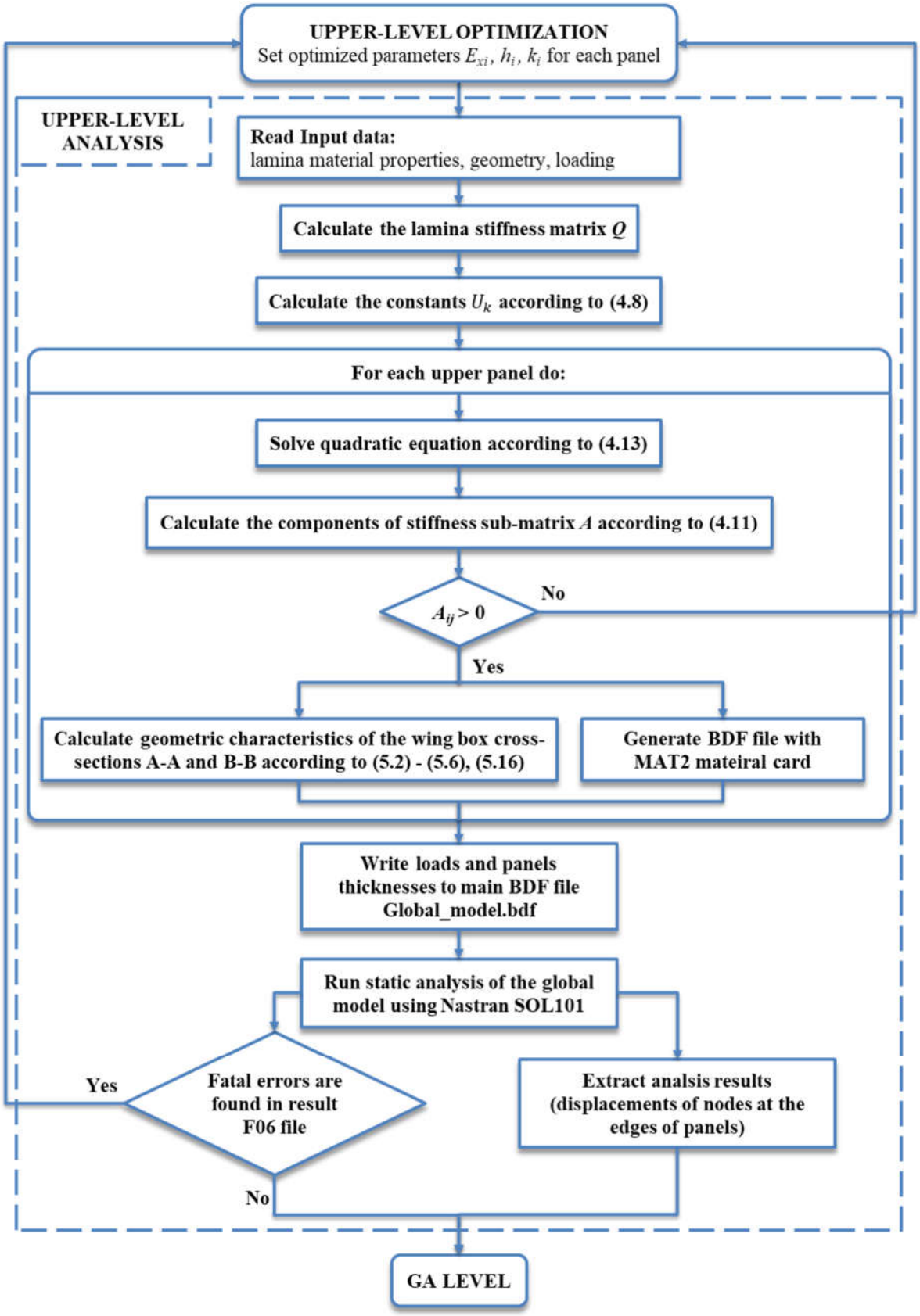

Fig. 5.7 Flow chart of the upper-level analysis

5.1.5 Description of the upper-level optimization workflow

The workflow for the upper-level optimization is shown in Fig. 5.8. At this level the following steps are performed:

1. Set upper-level parameters E_{xi}, h_i and k_i for each panel.
2. Perform upper-level analysis (see Subsection 5.1.4) in parallel for all generated points (experiments).
3. The parameters E_x , h and results of upper-level analysis (displacements of nodes at the edges of panels, structural stiffnesses EJ_x , EJ_y , GJ_z for cross-sections A-A and B-B) are transferred to the GA level (see Subsection 5.2).

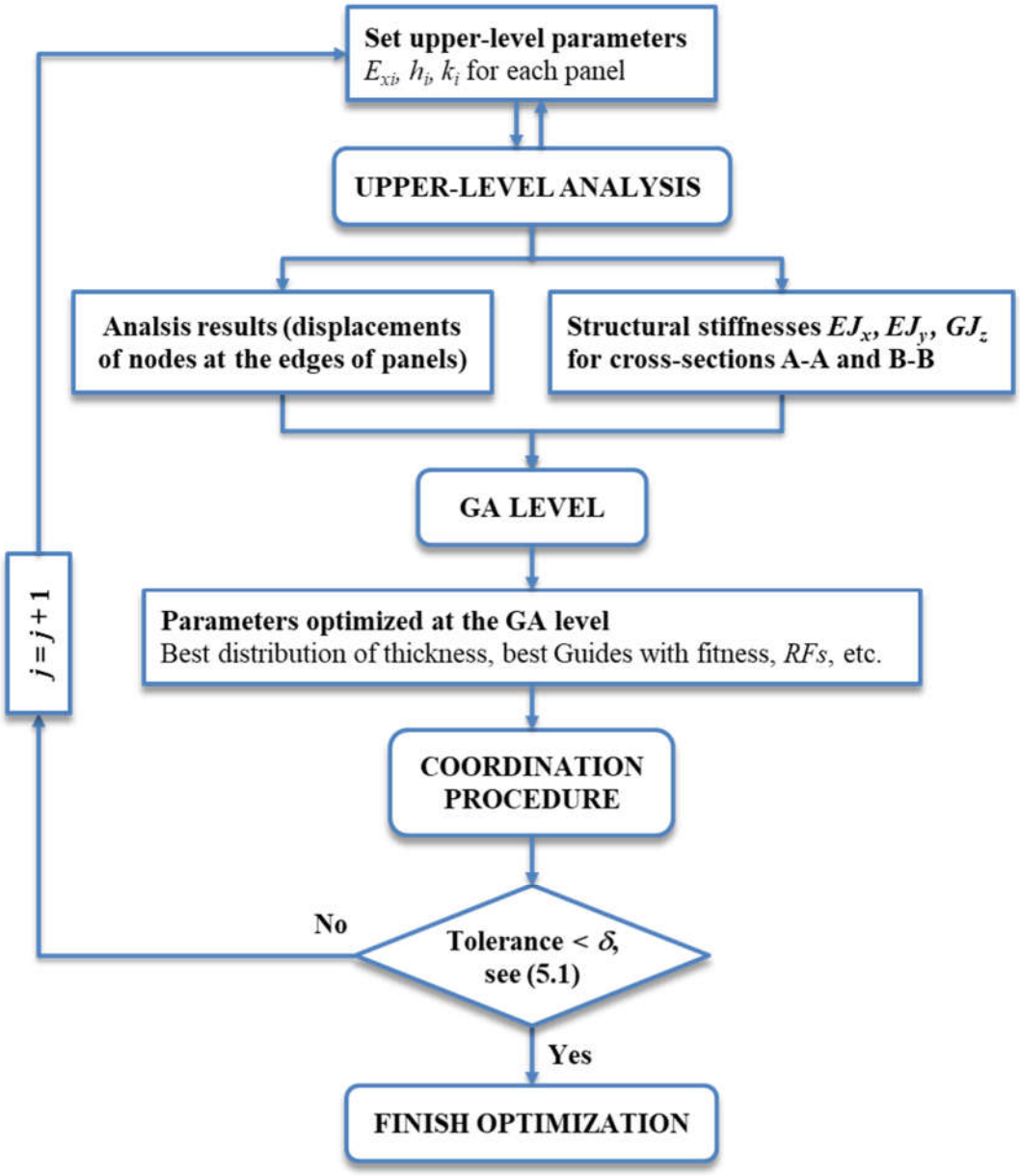

Fig. 5.8 Flow chart of the upper-level optimization

4. After the GA level is finished, a text file (formatting of the file see in Fig. 5.9, the guides in the file are sorted in ascending order of their fitness values) with best guides is transferred to the upper level. These guides are accompanied with the best thickness distribution, fitness values for all panels calculated according to (4.22) and minimum RFs of the panels. The fitness values (4.18) of the guides are also contained in this file.
5. The information about the best guides is transferred to the coordination procedure (see Subsection 5.4).
6. After the coordination procedure is finished the upper-level stop criterion (5.1) is checked.
7. If the tolerance is reached, meaning if the new point is too close to the already known points, the optimization is finished, otherwise new set of experiments is generated and the cycle 1 – 7 is repeated for all these experiments.

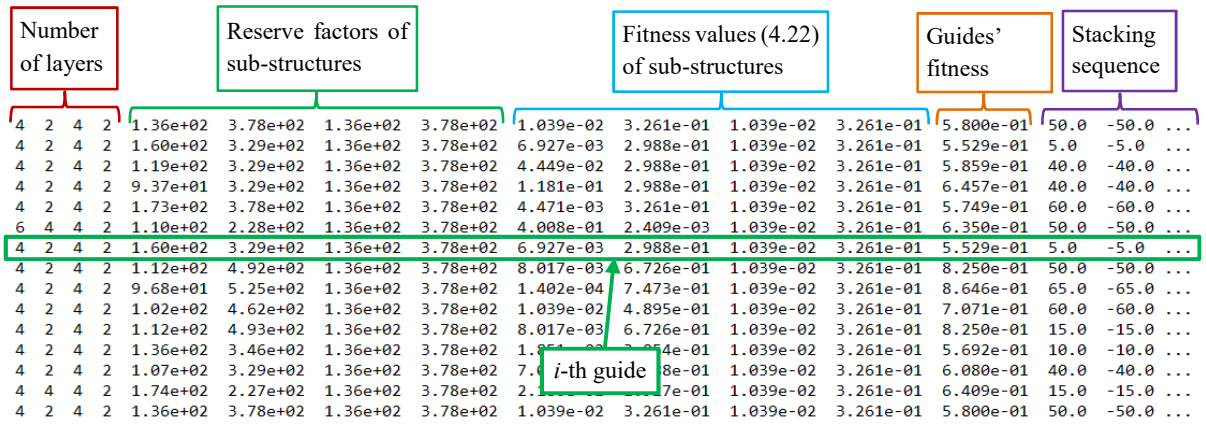

Fig. 5.9 Formatting of a "List_of_Guides.txt" file

5.2 GA-level optimization

The GA level deals with generation of guides and the GA, which performs genetic operations with the generated guides to improve them with help of simplified genetic evolution. No FE analysis is performed here, therefore there is no need in any FE model at this level. The fitness values (4.19) of the guides are calculated at this level. The algorithm developed for this level consists of three scripts, which were programmed using Python language ("Genetics_Guides_Opt.py", "Guides_Operator.py" and "Replacer.py" – see listings of the files in Appendix B). This algorithm has two modes, which are set by the "mode" parameter and the user can choose whatever he wants.

The first mode is "GenerateGuides". In this mode at the beginning the algorithm randomly generates the stacking sequences for the first generation of guides and writes them into the file "List_of_Guides.txt" (an example of the file formatting for a structure divided into 4 domains is shown in Fig. 5.9). The mode is usually used within normal optimization process when the upper level automatically calls the GA level at each optimization step.

The second mode is "ReadGuides". In this mode the first generation of guides is not generated, but read from the "List_of_Guides.txt" file, which should be provided by the user. The mode is intended for the case, when the user requires to improve the results of a chosen experiment by additional iterations of GA at the GA level, starting from the guides found at the last step during the optimization. This mode can be also used by the user in case if he wants to start optimization at the GA level with his own stacking sequences.

The flow chart of the GA level optimization is shown in Fig. 5.10. The following steps are performed:

1. If parameter fail = 0 (shows if the upper level analysis was successful) start the optimization, otherwise return to the upper level.
2. Check the "mode" parameter:
   a. If mode = "ReadGuides" the guides are read from the "List_of_Guides.txt" file (given by the user),
   b. Otherwise the first population of guides is generated by the "Guides_Operator.py" and written into the „List_of_Guides.txt" file.
3. Genetic operations (Selection, Crossover and Mutation - see Subsection 4.2) are performed with the guides. As the result of the operations two new guides appear.
4. The guides (the first generation of guides obtained at step 2.b or new guides obtained at step b) are transferred to the lower level (described below in Subsection 5.3), where they are optimized.
5. The fitness values (4.19) are calculated for the guides. If a guide has the reserve factor (calculated at the lower level) less than 1 for at least one panel, the fitness value of the guide is set to very high number, e.g., 10^{100}.
6. For the second and subsequent generations of guides the new guides replace the worst ones in the population using "Replacer.py". Thus, the next generation is created. The new generation of guides is written into „List_of_Guides.txt" file. The best guide of the population at each generation is saved into the „Best_guides.txt" file.
7. The stop criteria of the algorithm (see Subsection 4.2) are checked.
8. If at least one of the used criteria is met, the GA-level optimization is finished and the algorithm is stopped, otherwise the algorithm goes to step 3.
9. If the algorithm is stopped, "Best_Guides.txt" file is transferred to the upper level.

The algorithm can be customized using the following options (see Fig. 5.11):

- „Number of genes" corresponds to the half of the maximum number of layers in optimized structure,
- „Population size" – the number of individuals in any population, which corresponds to the number of guides in the „List_of_Guides.txt" file,
- „Stop Criterion" – the user can choose, which stop criterion or combination of them (see Subsection 4.2) to use,

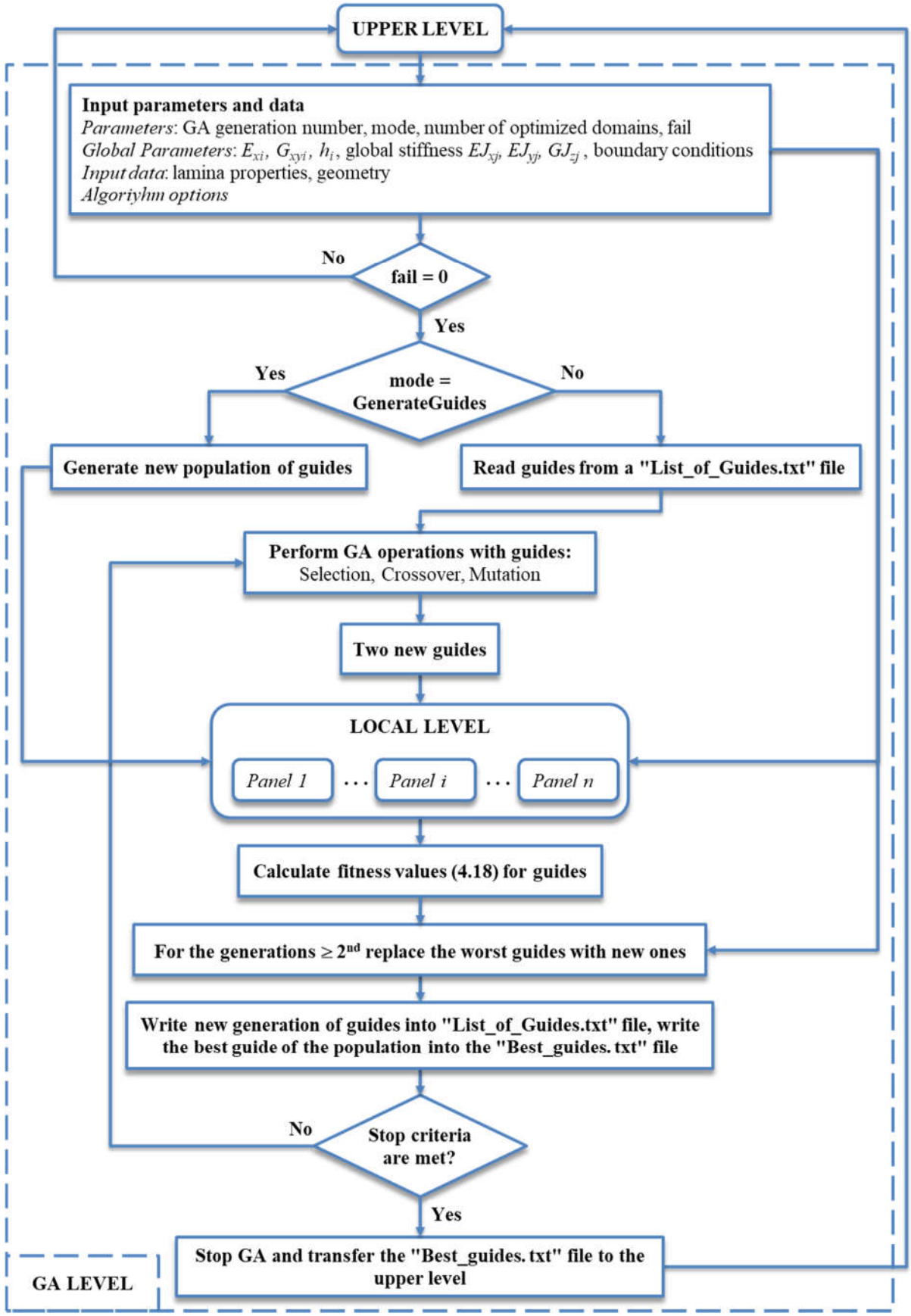

Fig. 5.10 Flow chart of the GA-level optimization

- „Fitness Tolerance Value“ - $F_{obj\,max}$ value for limiting the fitness value of the best guide (see Subsection 4.2),
- “Best Guides Number” – the number N_{best} in (4.21),
- “Best Guides Scatter Value, %” – ν value in (4.21),
- The last two options are required to connect the GA level with the lower level.

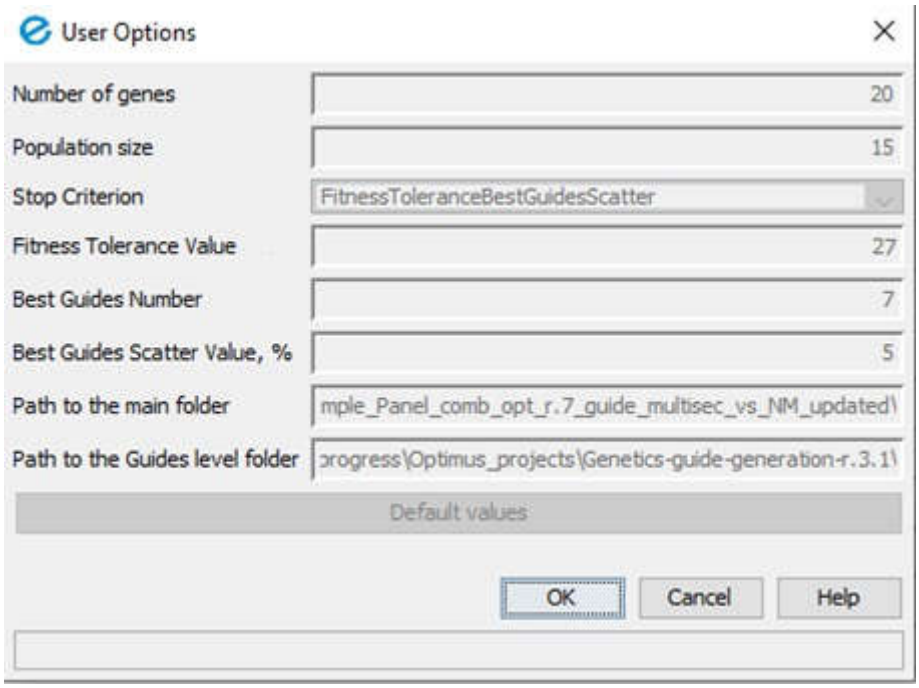

Fig. 5.11 Options for the genetic algorithm

5.3 Lower-level optimization

The lower level deals with optimization of the guides generated at the GA level. For each guide sent to the lower level the optimal distribution of thickness is found. It is done by successive optimization of each panel (see Fig. 5.2). For each panel the optimum thickness is calculated, which corresponds to the best value of the objective function (4.22). The minimum reserve factor (RF_{min}) is calculated for each panel based on the element strains and buckling load factor extracted from the local FE model buckling analysis and taking into account the best thickness and the stacking sequence of a particular guide. The RF_{min} is calculated as follows:

$$RF_{min} = min\left(\frac{\varepsilon_{1a}}{\varepsilon_1}; \frac{\varepsilon_{2a}}{\varepsilon_2}; \frac{\gamma_{12a}}{\gamma_{12}}; buckling\ load\ factor\right), \tag{5.17}$$

where the first three members in brackets correspond to reserve factors calculated according to maximum strain criterion ($\varepsilon_1, \varepsilon_2$ and γ_{12} are extracted from the FEA result file, $\varepsilon_{1a}, \varepsilon_{2a}$ and γ_{12a} are taken from Tab. 5.1), buckling load factor – first positive eigenvalue extracted from the FEA result file.

Before starting the lower-level optimization workflow, the initial local FE model should be built. It should be a detailed model. In general, when the geometry of the structure is

complex, there should be as many local models as the global model was divided into. Therefore, for each of them a lower-level optimization workflow should be build. In case of the example problem the panels being optimized have the same geometry. Thus, the only one initial local FE model (see *Fig. 5.12*) and corresponding to it lower-level optimization workflow were built (see Fig. 5.13).

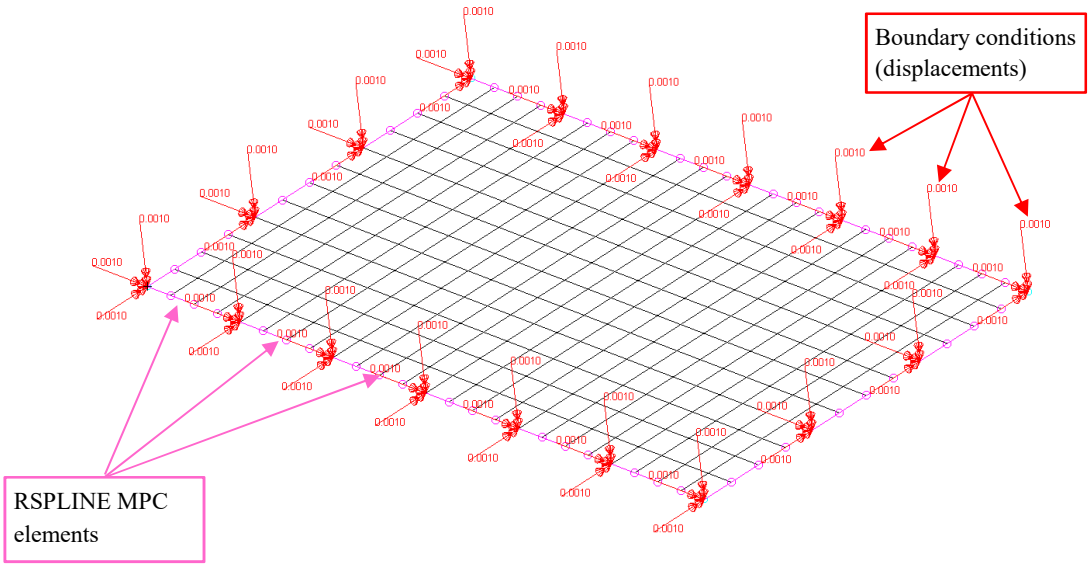

Fig. 5.12 Local FE model

The local model is created using MSC Patran preprocessor. Its geometry corresponds to the geometry of an optimized panel (see Fig. 5.1 and Fig. 5.2). The thickness of the panel is constant. The mesh is 16 times finer than in the global model. The boundary conditions are defined by the translational and rotational displacements taken from the global model nodes lying on the edges of the optimized panels (red arrows in the figure). The local model has 4 times more nodes on their edges than the global model. The displacements from the nodes of the global model can be applied only to the corresponding nodes of the local model. Therefore, the boundary conditions for the other nodes on the edges of the local model were interpolated using RSPLINE MPCs (multi-point constraints – see [43]) – pink elements on the edges of the panel (*Fig. 5.12*).

The laminate material is defined using PCOMP and MAT8 cards (see Appendix A). The PCOMP card is generated into an external BDF file by the "PCOMP_D_writer.py" Python script (see Appendix B) and included into the main file of the model using the INCLUDE statement. The initial stacking sequence defined in the initial FE model can be whatever because it will be replaced by "PCOMP_D_writer.py" script during optimization. The same is concerned to the displacements applied to the panel. The buckling FE analyses of the model are performed by MSC Nastran (SOL105) during the optimization at the lower level.

The lower-level optimization workflow shown in Fig. 5.13 performs the following steps for *i*-th panel (the listing of the main script "Genetics_vs_gold_parallel.py" is shown in Appendix B):

1. Upper-level parameters (E_{xi} , G_{xyi} , h_i), boundary conditions and guides from the "List_of_Guides.txt" (see Fig. 5.9) file transferred from the GA level are read.
2. The search interval $[2, M]$ for number of leyers is set, where $M = N/2, N$ is the maximum number of layers defined by the user. Two new points are calculated within the interval using "golden ratio": $n_1 = M - (M-2)/\Phi$, $n_2 = 2 + (M-2)/\Phi$, where $\Phi = 1.618$. Thus, the numbers $(2, n_1, n_2, M)$ are the starting points (experiments) of the algorithm. For each guide the search starts from these experiments.
3. The next operations for each guide and each generated experiment are done:
   a. BDF file with PCOMP material card for the local FE model is generated,
   b. E_x and G_{xy} moduli of laminates are calculated,
   c. The thickness h of the panel is calculated,
   d. The values hE_x and hG_{xy} are calculated,
   e. The fitness value (4.22) for the panel is calculated.
4. Buckling analyses of the local FE model using MSC Nastarn (SOL105) are performed in parallel for all generated above experiments.
5. Structural responses (lamina strains for all elements $\varepsilon_1, \varepsilon_2, \gamma_{12}$ and the first positive eigenvalue) are extracted from the result F06 files.
6. The minimum reserve factor RF_{min} (5.17) of the panel is calculated for each design.
7. The information about each design (number of layers, fitness value and RF_{min}) is written to the storage-arrays. Each guide has its own array. These arrays store the information about guides ever been estimated within a particular lower-level optimization.
8. The stop criteria are checked for each guide. There are two criteria:
   a. For each guide – if the tolerance is reached, meaning if the new point is too close to the already known points, then stop optimizing a particular guide and lock it, otherwise generate new points for the guide according to the "Golden Section" method and go to step 3.
   b. For all guides – if all guides are locked, then stop the main optimization cycle for all guides and go to step 9, otherwise go to step 3.
9. The storage-arrays are sorted in fitness ascending order. Thus, the first components of the arrays have the smallest fitness values. Let's call them "temporary-minimum". Remember this minimum.
10. The storage-arrays are sorted in number-of-layers ascending order. This means the thinnest designs are moved to the first components of the arrays.
11. Checks if the designs close to the temporary-minimum were simulated also. If no, generate new points corresponding to these designs and go to the step 3, otherwise go to step 11.
12. The storage-arrays are sorted in fitness ascending order like in the step 9.
13. Starting from the first components the storage-arrays are checked for the condition if the designs stored there have $RF_{min} \leq 1$. If yes, these designs are deleted because they are failed, otherwise go to step 14.

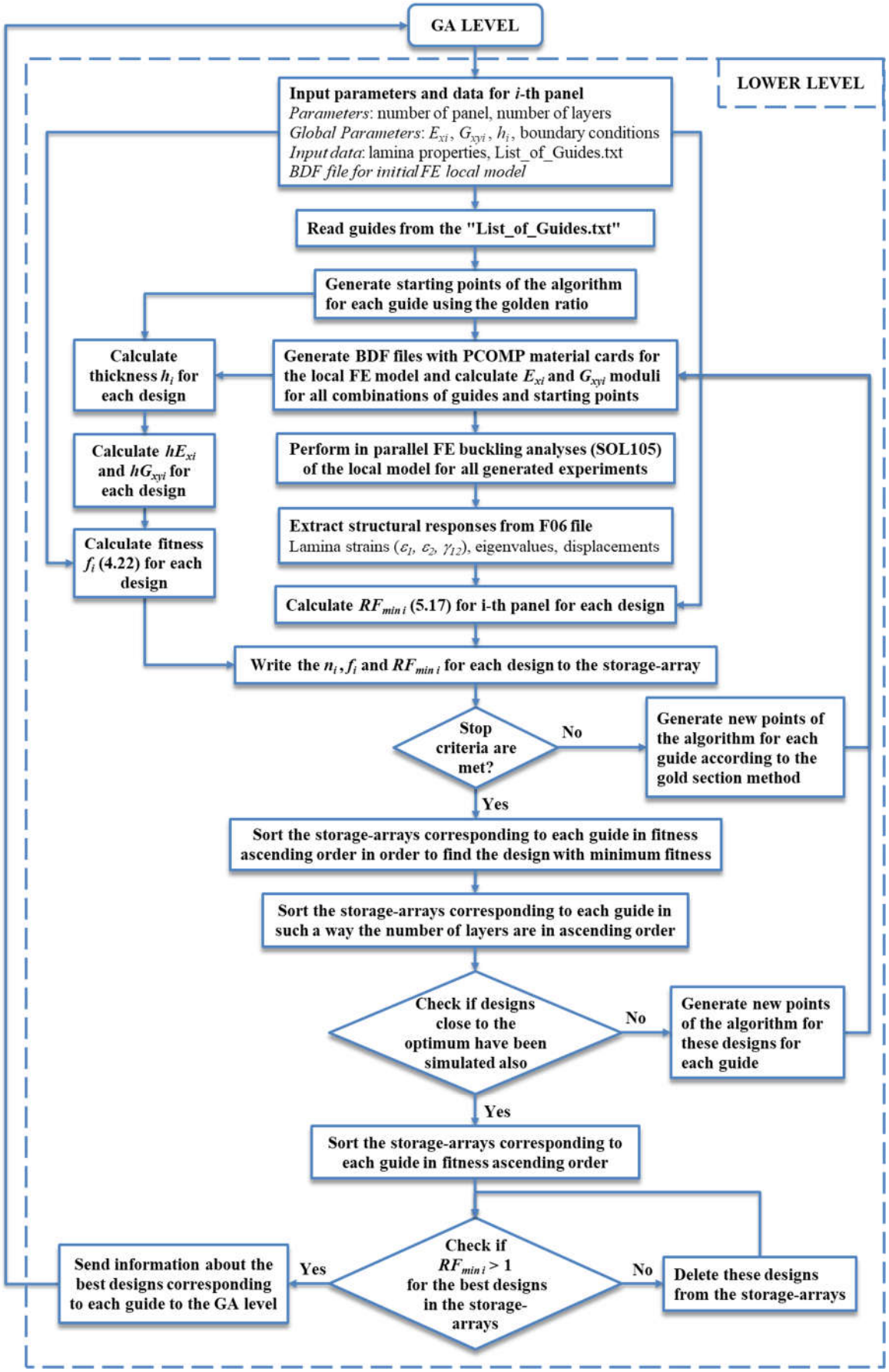

Fig. 5.13 Flow chart of the lower-level optimization

14. As soon as a design with $RF_{min} > 1$ is found the algorithm is stopped and this design is taken as the optimal one.
15. The information about optimal designs found for each guide (number of layers, fitness value and RF_{min}) is sent to the GA level. The workflow is finished and the focus is passed to the GA level.

The workflow is performed for each panel one by one.

5.4 Coordination procedure

Before building the coordination procedure workflow the initial "detailed" global FE model should be build. This model should take into account the stacking sequences of the best guides and their best thickness distribution found at the GA level. These stacking sequences are modeled using PCOMP cards (see Appendix A). There are four PCOMP cards in the "detailed" global model. Each of them corresponds to one of the four optimized panels of the upper skin. Thus, the laminates of the panels are not modeled using MAT2 cards (see Appendix A) like in Subsection 5.1.4 above.

The workflow for the coordination procedure is shown in Fig. 5.14. The following steps are performed:

1. A text file (see Subsection 5.2) with the best guides is transferred from the GA level. This file contains stacking sequences of the guides, distribution of thicknesses (number of layers) through the upper skin, minimum RFs for each panel, fitness values (4.22) for each panel and fitness values (4.18) for each guide. The guides in the file are sorted in ascending order of their fitness values.
2. Starting from the beginning of the file the guides are read one by one and the next operations are repeated until the stop criterion is reached:
    a. A BDF file with PCOMP material card is generated for each optimized panel using the "PCOMP_global_writer.py" Python script (see Appendix B). These files are generated based on the information about the chosen guide.
    b. Buckling analysis of the "detailed" global FE model is performed using MSC Nastran (SOL105) and the BDF files with PCOMP cards generated in the previous step.
    c. Required structural responses are extracted from the result F06 file (maximum lamina strains $\varepsilon_1, \varepsilon_2, \gamma_{12}$ for all elements and displacements for all nodes).
    d. Minimum reserve factor RF_{min} (5.17) is calculated for the entire structure using the extracted responses and lamina material properties:
    e. If $RF_{min} > 1$, the stop criterion is met and the cycle is stopped, the algorithm goes to step f, otherwise the guide is rejected.
    f. If all guides from the file were rejected, the entire experiment is treated as failed and is not used in further upper-level optimization steps.
3. The weight of the upper skin is calculated and the experiment is used in further optimization steps.

4. The focus is passed to the upper-level optimization.

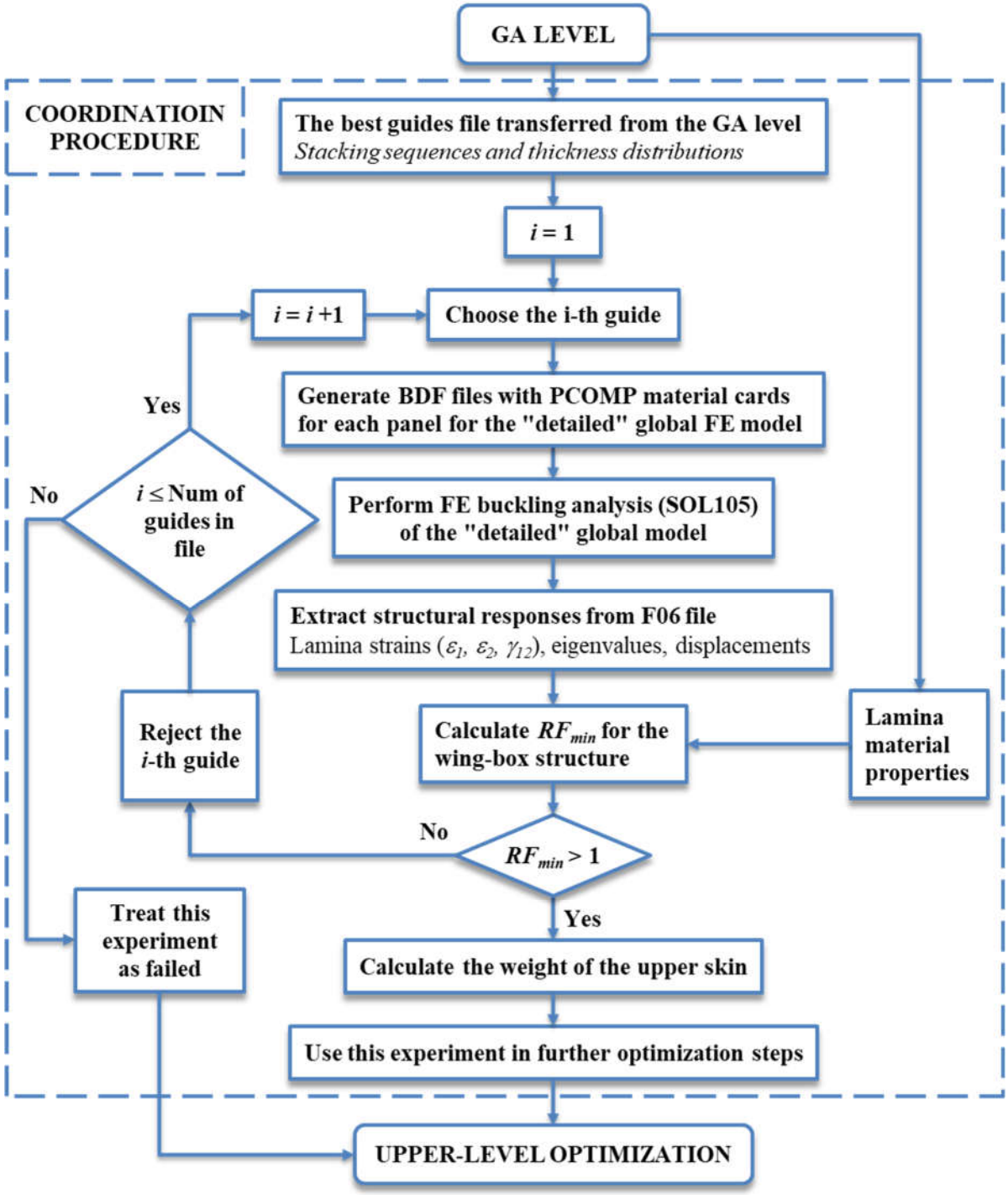

Fig. 5.14 Flow chart of the coordination procedure

5.5 Optimization results for the example problem

The example problem was solved on two computers with Intel Xenon CPU E5-2620 2.00 GHz (6 physical cores and 12 threats) connected in a network (16 processes are allowed by the Optimus license to be run in parallel). For comparison, the authors in [23] used 200 node (400 processor) 1.4 GHz dual Opteron cluster. For solving the same problem they used 51 processor. They reported the average run time of 12 hours. The algorithm has found the optimal solution

with the weight of 12.018 kg (10 layers for panel 1, 11 layers for panel 2, 6 layers for panel 3 and 7 layers for panel 4). The stacking sequence of the optimal guide was the following:

$$[90°/45°/90°/-45°/45°/90°_2/-45°/0°/90°/\pm45°/0°_3/45°/0°/-45°/0°/90°].$$

In the present work the optimization algorithms at upper and GA levels have their options set to the values shown in Tab. 5.3.

Tab. 5.3 Values for the algorithms options

Option	**Value**
GA level	
Mode	GenerateGuides
Number of genes	20
Population size	15
Stop criterion	TitnessTolerance + GuidesScatter
Max number of generations	20
Fitness tolerance value	20
Best guides number	7
Best guides scatter value, %	5
Upper level	
Number experiments for first LH	15
Random seed	15-77
Maximum number of evaluations	30
Tolerance	0.1
Model type	RBF
Spline type	Cubic
Optimization method	Min. f-x.std
Number of Sigma	3-5

The orientation angles allowed for generation of guides were the following 0°, 15°, 30°, 45°, 60°, 75°, and 90°. The results of 7 subsequent runs of the entire optimization algorithm are shown in Tab. 5.4. The maximum number of iterations allowed to the upper level algorithm was 30 (see the column "# of iterations"). However, this does not mean the algorithm went through all optimization levels at each iteration. Since not every combination of upper-level parameters gives real and positive definite stiffness matrices of laminates, some iterations were cancelled by the upper-level algorithm. Thus, there is another column named "# of full

iterations“ in the table. It shows the number of iterations fully completed including all optimizaiton levels. Therefore, the muximum number of full iterations required to find the best solution was 14. The average run time was about 25 hours. The only one found optimal solution is slightly heavier than that found in [23]. Moreover, 3 found solutions are even lighter than than that found in [23]. The stacking sequences of the optimal guides corresponding to the solusions listed in Tab. 5.4 are shown in Tab. 5.5. The plies of each desing for each panel should be taken from the end of the guide.

Tab. 5.4 The optimal solutions found during several runs

#	Duration, hh:mm	Weight, kg	# of iterations	# of full iterations	# of plies for *i*-th panel:			
					1	2	3	4
1	***26:57***	***10.604***	***30***	***14***	***10***	***8***	***6***	***6***
2	19:32	11.311	19	8	8	10	6	8
3	***31:27***	***10.604***	***15***	***4***	***8***	***8***	***10***	***4***
4	27:21	12.725	15	1	8	12	6	10
5	20:12	11.311	30	12	8	6	8	10
6	***29:15***	***10.604***	***30***	***13***	***10***	***8***	***6***	***6***
7	21:11	11.311	27	13	10	8	8	6

Tab. 5.5 The stacking sequences of the found optimal guides

#	Guide’s stacking sequence
1	***[±45°/±75°/90°/0°/90°/0°/±45°/±15°/90°/0°/±60°/±15°$_2$]***
2	[90°$_2$/0°/90°/±15°/90°$_2$/±15°/±75°/0°/90°/0°/90°/0°/90°$_3$]
3	***[90°/0°/±45°/0°$_2$/±45°/±15°/±75°/±30°$_2$/±60°/0°/90°]***
4	[±15°/90°$_2$/±75°/±15°/90°/0°/±60°/90°$_2$/±30°/90°$_2$/±75°]
5	[±75°/±60°/90°$_2$/±75°/±30°/±15°/±60°$_3$/±45°]
6	***[90°/0°/±45°/0°$_2$/±45°/±15°/±75°/±30°$_2$/±60°/0°/90°]***
7	[±75°/0°$_2$/±30°/±75°/0°$_3$/90°/0°$_3$/±15°/90°$_2$/±45°]

6 Implementation of multi-objective optimization

So far it has been about single-objective optimization including the above described applicaiton of the methodology. However, the topic of the present work declares the multi-objective optimization.

By virtue of the Optimus software it is possible to add several more objectives to the uppe-level optimization very easily. Optimus has both many built-in methods for multi-objective optimization and instruments for the optimization results postprocessing [42].

For example, the simplest multi-objective optimization method is weighted objective method. It adds all the objective functions together using different weighting coefficients for each, so that the multiobjective optimization problem is transformed to a single-objective optimization problem by creating one function of the form:

$$f(x) = \sum_{i=1}^{k} \omega_i f_i(x), \tag{6.1}$$

where $\omega_i \geq 0$ - weighting coefficients, $\sum_{i=1}^{k} \omega_i = 1$, $f_i(x)$ - objective functions.

The Pareto front can be obtained by trying different set of weighting coefficients.

7 Conclusions

An optimization methodology for thin-walled composite structures was developed during the present work. The methodology was realized based on commercial optimization software Noesis Optimus, originally programmed optimization code and commercial FEA software MSC Nastran. It fulfills all the objectives of the thesis (see Section 3). Specifically the developed methodology can be applied for optimization of a complex composite structure with variable stiffness, meaning the structure can be divided into several sub-structures with different stiffness. Each sub-structure may, in general, consist of several skin panels of different thicknesses, spars and stringers and separated from the neighboring sub-structures with ribs or frames. Several load cases are also possible.

The optimization parameters are thicknesses and stacking sequences of the optimized structural elements as it was planned. However during the example problem optimization the only objective was weight minimization of the empty structure, the Optimus software allows simply and quickly add several another objectives, e.g. mimizaiton of maximium deflection, maximum twist angle, etc. The optimization constrains were strength and buckling of the structure and continuity of the neighboring sub-structures. Other constrains could be easily added. Then a malti-objective optimization method can be chosen and applied from those provided by the Optimus.

The developed methodology conforms to the basic aspects of a modern optimization methodology recited in the state-of-the-art review (see Section 1).

The methodology was applied to solve an example problem where the weight of a simple wing box was minimized. The results has shown the methodology is very effective. It allows finding the lighter designs in less time or using less computational resources in comparison to the previously developed methodologies, e.g. [23].

The novelties of the present thesis are enclosed in the next aspects:

- the general approach to the problem decomposition,
- the approach to the definition of a laminate through E_x, k and h parameters (see Subsection 4.1), which allows to simply distribute composite material through the structure without insight into its details (stacking sequence, orientation angles, etc.),
- the newly developed genetic algorithm (see Subsections 4.2 and 5.2),
- the newly developed parallel one-dimensional algorithm based on “Golden Section” method (see Subsections 4.3 and 5.3).

Besides the mentioned, the methodology has the next advantages.

Flexibility. It can be applied for optimization of a very wide class of composite structures. Within this thesis a FEA software was used for performing structural analyses. However, the flexibility of the developed interface allows using any other analysis methods, e.g., analytical calculations, surrogate models, etc., which can be simply integrated instead of FEA. A combination of FEA and other analysis methods can be used also. In case if FEA software is

used for performing structural analyses the shape, complexity and loading of the structure can be whatever. The methodology allows to set many types of objectives and apply many types of constraints.

Simplicity. The developed interface is very intuitive and can be used by an engineer on-the-fly without a special time and coast consuming training.

Parallel processing. The analyses of models are run in parallel.

Of course, the methodology and the interface are not ideal and have disadvantages. They are mentioned below.

Commercial software Optimus. To integrate all levels, algorithm, and software a commercial optimization software Noesis Optimus is used which requires a license. However, a domestic interface can be developed easily, since the main and auxiliary optimization code is programmed by the author using Python language.

Computational expensiveness. Since a GA is used at the GA level, the methodology requires extensive computational resources. When FE software is used for structural analyses, it is even more computationally expensive.

LIST OF DESIGNATIONS AND ABBREVIATIONS

A – the membrane sub-matrix of the laminate stiffness matrix, Pa m,

$\bar{A}$ - normalized membrane stiffness sub-matrix, Pa,

AE_z – extensional (longitudinal) stiffness of a cross-section, N,

C_f – number of groups of individuals, which are candidates to be replaced in the replacement operator of a GA,

D – bending sub-matrix of the laminate stiffness matrix, Pa m^3 ,

E_{1i}, E_{2i} –Young's moduli of the i-th layer in longitudinal and transverse directions, Pa,

E_x, E_y – longitudinal and transversal Young's moduli of a laminate, Pa,

EI_z, EI_y – first moments of a cross-section area, N m,

EJ_z , EJ_y – bending cross-sectional stiffnesses, N m^2,

F, F_{obj} – objective function of an upper-level optimization,

$\bar{F}_{obj}$ – average objective function (fitness) value for N_{best} guides,

f – objective function of a lower-level optimization,

G_{12i} – shear modulus of the i-th layer, Pa,

G_{xy} – in-plane shear modulus of a laminate, Pa,

GJ – torsional cross-sectional stiffness, N m^2,

h - total thickness of a laminate, m,

h_0 – thicknesses of a lamina, m,

h_i – thicknesses of layers with the same orientation, m,

k – ratio $\frac{E_x}{E_y}$,

N_s – total number of defined cross sections of a structure,

N_{best} – number of best guides within an actual population,

N_{tot} – total number of guides in a population,

P – penalty factor for penalizing an objective function,

P_c – crossover operator probability in a GA,

P_m – mutation operator probability in a GA,

Q – lamina stiffness matrix, Pa,

RF_j – reserve factors of different failure modes, including buckling ones,

S – number of individuals in each group of candidates to be replaced in the replacement operator of a GA,

U_i - constants depending on the lamina stiffness matrix, Pa,

W – total weight of the structure or a structural element, kg,

$\bar{X}$ – vector of variables (number of layers, stacking sequence, geometric dimensions, etc.);

x_c, y_c – coordinates of the cross-section's centroid, m,

$\varepsilon_{1a}, \varepsilon_{2a}, \gamma_{12a}$ – linear and shear allowable strains of a lamina material,

$\varepsilon_1, \varepsilon_2, \gamma_{12}$ – linear and shear strains of a lamina in the local coordinate frame,

θ_k – orientation angle of the k-th layer of a laminate, degrees,

μ_{1i}, μ_{2i} – Poisson's coefficients of the i-th layer,

μ_{xy}, μ_{yx} – Poison's coefficients of a laminate,

ξ_k – lamination parameters (functions depending on the orientation of laminate's layers, their thickness and stacking sequence), m,

$\bar{\xi}_k$ – normalized lamination parameters,

ρ –density of a lamina material, kg/m^3,

$\Phi = 1.618$ – Golden ratio,

CAD – Computer Aided Design,

DOF – Degree of Freedom,

EGO - Efficient Global Optimization method,

FE(A) – Finite Element (Analysis),

GA – Genetic Algorithm,

MDO – Multi-Disciplinary Optimization,

MPC - Multi-Point Constraint,

PCL – Patran Command Language,

SQP – Sequential Quadratic Programming.

REFERENCES

[1] SOBIESZCZANSKI-SOBIESKI, J., B.B. JAMES and A.R. DOVI. 1985. Structural optimization by multilevel decomposition. *AIAA Journal*. **23**(11), 1775-1782.

[2] WALSH, J.L., K.C. YOUNG, J.I. PRITCHARD, H.M. ADELMAN and W.R. MANTAY. 1995. *NASA Technical Paper 3465: Integrated aerodynamic/dynamic/ structural optimization of helicopter rotor blades using multilevel decomposition*. Hampton, Virginia: Langley Research Center, National Aeronautic and Spase Administration.

[3] SCHMIT, L.A. and M. MEHRINFAR. 1982. Multilevel optimum design of structures with fiber-composite stiffened-panel components. *AIAA Journal*. **20**(1), 138–147.

[4] GASBARRI, P., L.D. CHIWIACOWSKY and H.F. de CAMPOS VELHO. 2010. A hybrid multilevel approach for aeroelastic optimization of composite wing-box. *Structural and Multidisciplinary Optimization*. **39**(6), 607-624.

[5] DUVAUT, G., G. TERREL, F. LÉNÉ and V.E. VERIJENKO. 2000. Optimization of fiber reinforced composites. Composite Structures. **48**(1-3), 83-89.

[6] KAM, T. and J.A. SNYMAN. 1989. Optimal design of laminated composite plates with dynamic and static considerations. *Computers & Structures*. **32**(2), 387-393.

[7] KAM, T. and M.D. LAI. 1989. Multilevel optimal design of laminated composite plate structures. Computers & Structures. **31**(2), 197-202.

[8] WATKINS, R.I. and A.J. MORRIS. 1987. A multicriteria objective function optimization scheme for laminated composites for use in multilevel structural optimization schemes. *Computer Methods in Applied Mechanics and Engineering*. **60**(2), 233-251.

[9] GRIHON, S., L. KROG and D. BASSIR. 2009. Numerical Optimization applied to structure sizing at AIRBUS: A multi-step process. In: *International Journal for Simulation and Multidisciplinary Design Optimization*. **3**(4), p. 432-442.

[10] GHIASI, H., D. PASINI and L. LESSARD. 2009. Optimum stacking sequence design of composite materials Part I: Constant stiffness design. *Composite Structures*. **90**(1), 1-11.

[11] ZEHNDER, N. and P. ERMANNI. 2006. A methodology for the global optimization of laminated composite structures. *Composite Structures*. **72**(3), 311-320.

[12] ZEHNDER, N. and P. ERMANNI. 2007. Optimizing the shape and placement of patches of reinforcement fibers. *Composite Structures*. **77**(1), 1-9.

[13] KIM, J.-S., C.-G. KIM and C.-S. HONG. 1999. Optimum design of composite structures with ply drop using genetic algorithm and expert system shell. *Composite Structures*. **46**(2), 171-187.

[14] ZABINSKY, Z.B., M.E. TUTTLE and C. KHOMPATRAPORN. 2006. A Case Study: Composite Structure Design Optimization. In: *Global Optimization*. Springer US, p. 507-528. Nonconvex Optimization and Its Applications.

[15] TATTING, B. and Z. GÜRDAL. 2001. Analysis and design of tow-steered variable stiffness composite laminates. In: *American Helicopter Society: Structure Specialists' Meeting*. Williamsburg, VA.

[16] HUANG, J. and R.T. HAFTKA. 2005. Optimization of fiber orientations near a hole for increased load-carrying capacity of composite laminates. *Structural and Multidisciplinary Optimization*. **30**(5), 335-341.

[17] GUO, S. 2007. Aeroelastic optimization of an aerobatic aircraft wing structure. *Aerospace Science and Technology*. **11**(5), 396-404.

[18] ZHAO, Q., Y. DING and H. JIN. 2011. A Layout Optimization Method of Composite Wing Structures Based on Carrying Efficiency Criterion. *Chinese Journal of Aeronautics*. **24**(4), 425-433.

[19] HAILIAN, Y. and Y. XIONGQING. 2010. Integration of Manufacturing Cost into Structural Optimization of Composite Wings. *Chinese Journal of Aeronautics*. **23**(6), 670-676.

[20] DEB, K., A. PRATAP, S. AGARWAL and T. MEYARIVAN. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*. **6**(2), 182-197.

[21] SERESTA, O., Z. GÜRDAL, D.B. ADAMS and L.T. WATSON. 2007. Optimal design of composite wing structures with blended laminates. *Composites Part B: Engineering*. **38**(4), 469-480.

[22] LIU, B., R. HAFTKA and M. AKGUN. 1998. Composite wing structural optimization using genetic algorithms and response surfaces. In: *7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*. Reston, Virigina: American Institute of Aeronautics and Astronautics, p. 1-12.

[23] ADAMS, D., L.T. WATSON, O. SERESTA and Z. GÜRDAL. 2007. Global/Local Iteration for Blended Composite Laminate Panel Structure Optimization Subproblems. *Mechanics of Advanced Materials and Structures*. **14**(2), 139-150.

[24] GANGULI, R. 2013. Optimal Design of Composite Structures: A Historical Review. *Journal of the Indian Institute of Science*. **93**(4), 557-570.

[25] AWAD, Z.K., T. ARAVINTHAN, Y. ZHUGE and F. GONZALEZ. 2012. A review of optimization techniques used in the design of fibre composite structures for civil engineering applications. *Materials & Design*. **33**, 534-544.

[26] GHIASI, H., K. FAYAZBAKHSH, D. PASINI and L. LESSARD. 2010. Optimum stacking sequence design of composite materials Part II: Variable stiffness design. *Composite Structures*. **93**(1), 1-13.

[27] ADAMS, D.B., L.T. WATSON, Z. GÜRDAL and C.M. ANDERSON-COOK. 2004. Genetic algorithm optimization and blending of composite laminates by locally reducing laminate thickness: Variable stiffness design. *Advances in Engineering Software*. **35**(1), 35-43.

[28] JOAQUIM, R.M. and R.B. TIMOTHY. 2017. Enabling practical wing design via high-fidelity multidisciplinary optimization. In: *International Forum on Aeroelasticity and Structural Dynamics IFASD*. Como, Italy.

[29] MITCHELL, M. 1998. *An Introduction to Genetic Algorithms*. Cambridge, London: A Bradford Book MIT Press.

[30] KARABOGA, D. and B. BASTURK. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *Journal of Global Optimization*. **39**(3), 459-471.

[31] KARABOGA, D. and B. AKAY. 2009. A comparative study of Artificial Bee Colony algorithm. *Applied Mathematics and Computation*. **214**(1), 108-132.

[32] RAMA MOHAN RAO, A. 2009. Lay-up sequence design of laminate composite plates and a cylindrical skirt using ant colony optimization. In: Proceedings of the Institution of Mechanical Engineers, Part G: *Journal of Aerospace Engineering*. **223**(1), p. 1-18.

[33] AXINTE, A., L. BEJAN, N. TARANU and P. CIOBANU. 2013. Modern approaches on the optimization of composite structures. *The Bulletin of the Polytechnic Institute of Jassy: Construction, Architecture Section*. **59**(73), 43-54.

[34] SETOODEH, S., M.M. ABDALLA and Z. GÜRDAL. 2006. Design of variable–stiffness laminates using lamination parameters. *Composites Part B: Engineering*. **37**(4-5), 301-309.

[35] SETOODEH, S., Z. GÜRDAL and L.T. WATSON. 2006. Design of variable-stiffness composite layers using cellular automata. *Computer Methods in Applied Mechanics and Engineering*. **195**(9-12), 836-851.

[36] HUANG, J. and R.T. HAFTKA. 2005. Optimization of fiber orientations near a hole for increased load-carrying capacity of composite laminates. *Structural and Multidisciplinary Optimization*. **30**(5), 335-341.

[37] JING, Z., Q. SUN and V.V. SILBERSCHMIDT. 2016. A framework for design and optimization of tapered composite structures. Part I: From individual panel to global blending structure. *Composite Structures*. **154**, 106-128.

[38] LIU, D., V. TOROPOV, M. ZHOU, D. BARTON and O. QUERIN. 2010. Optimization of Blended Composite Wing Panels Using Smeared Stiffness Technique and Lamination Parameters. In: *51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference, Orlando, Florida, 18th AIAA/ASME/AHS Adaptive Structures Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics.

[39] SYMONOV, V. S., I. S. KARPOV and J. JURAČKA. 2013. Optimization of a Panelled Smooth Composite Shell with a Closed Cross-Sectional Contour by Using a Genetic Algorithm. *Mechanics of Composite Materials*. **49**(5), 563-570.

[40] ADAMS, D.B., L.T. WATSON and Z. GÜRDAL. 2003. Optimization and Blending of Composite Laminates Using Genetic Algorithms with Migration. *Mechanics of Advanced Materials and Structures*. **10**(3), 183-203.

[41] LIU, B. and R.T. HAFTKA. 2001. Composite wing structural design optimization with continuity constraints. In: *19th AIAA Applied Aerodynamics Conference*. Reston, Virigina: American Institute of Aeronautics and Astronautics, p. 1-12.

[42] *Optimus Theoretical Background*. 2019. Leuven, Belgium: Noesis Solutions.

[43] *MSC Nastran 2018: Quick Reference Guide*. 2018. Newport Beach, CA: MSC Software Corporation.

Appendix A Listing of BDF files

A.1 Two-dimensional orthotropic material (MAT8 card) [1]

The MAT8 material model is the simplest one and the most used in the engineering practice. The MAT8 entry is used to define a two-dimensional orthotropic stress-strain relationship as shown in eq. (A.1) and (A.2). The MAT8 entry can only be used with the plate and shell elements. Eq. (A.1) defines the in-plane stress-strain relationship. The transverse shear stress - transverse shear strain relationship is defined by eq. (A.2).

$$\begin{Bmatrix}\varepsilon_1\\ \varepsilon_2\\ \gamma_{12}\end{Bmatrix}=\begin{bmatrix}\frac{1}{E_1} & \frac{-\nu_{12}}{E_1} & 0\\ \frac{-\nu_{12}}{E_1} & \frac{1}{E_2} & 0\\ 0 & 0 & \frac{1}{G_{12}}\end{bmatrix}\begin{Bmatrix}\sigma_1\\ \sigma_2\\ \tau_{12}\end{Bmatrix}+(T-T_{REF})\begin{Bmatrix}\alpha_1\\ \alpha_2\\ 0\end{Bmatrix}, \tag{A.1}$$

where $\varepsilon_1, \varepsilon_2, \gamma_{12}$ – in-plane normal and shear strains, $\sigma_1, \sigma_2, \tau_{12}$ – in-plane normal and shear stresses.

$$\begin{Bmatrix}\tau_{1z}\\ \tau_{2z}\end{Bmatrix}=\begin{bmatrix}G_{1z} & 0\\ 0 & G_{2z}\end{bmatrix}\begin{Bmatrix}\gamma_{1z}\\ \gamma_{2z}\end{Bmatrix}, \tag{A.2}$$

where τ_{1z}, τ_{2z} – transverse shear stresses, G_{1z}, G_{2z} – transverse shear moduli, γ_{1z}, γ_{2z} – transverse shear strains.

The format of the Bulk Data entry for MAT8 is as shown in Tab. A.1 below.

Tab. A.1 MAT8 card

1	2	3	4	5	6	7	8	9	10
MAT8	MID	E_1	E_2	ν_{12}	G_{12}	G_{1z}	G_{2z}	RHO	
	A_1	A_2	T_{REF}	X_t	X_c	Y_t	Y_c	S	
	GE	F_{12}	STRN						

The fields in the Tab. A.1 have the next meaning:
MID - material identification number.
RHO - mass density.
X_t, X_c - allowable stresses or strains in tension and compression, respectively, in the longitudinal direction.
Y_t, Y_c - allowable stresses or strains in tension and compression, respectively, in the lateral direction.
S - allowable stress or strain for in-plane shear.
GE - structural damping coefficient.
STRN - request for the maximum strain theory only (see STRN in PCOMP/PCOMPG entry (see Subchapter A.3 below). Indicates whether X_t, X_c, Y_t, Y_c, and S are stress or strain allowable. (Real = 1.0 for strain allowable; blank (Default) for stress allowable).

Remarks:

- If G_{1z} and G_{2z} values are specified as zero or blank, then transverse shear flexibility calculations will not be performed, which is equivalent to zero shear flexibility (i.e., infinite shear stiffness). An approximate value for G_{1z} and G_{2z} is the in-plane shear modulus G_{12}. If test data are not available to accurately determine G_{1z} and G_{2z} for the material and transverse shear calculations are deemed essential; the value of G_{12} may be supplied for G_{1z} and G_{2z}.
- X_t, Y_t, and S are required for composite element failure calculations when requested in the FT (Failure theory/criterion used) field of the PCOMP/PCOMPG entry. X_c and Y_c are also used but not required.
- T_{REF} and GE are ignored if this entry is referenced by a PCOMP/PCOMPG entry.
- T_{REF} is used in SOL101 only as the reference temperature for the calculation of thermal loads. TEMPERATURE(INITIAL) may be used for this purpose, but T_{REF} must then be blank.

A.2 Two-dimensional anisotropic material (MAT2 card) [1]

The MAT2 entry is used to specify a general anisotropic two-dimensional stress-strain relationship of the form shown in eq. (A.3). In case if MAT2 is used to define the interlaminar properties of a composite material the eq. (A.4) is used. The MAT2 entry can only be used with plate and shell elements. The reference temperature is given by T_{REF} and the thermal expansion coefficients are α_1, α_2, and α_3. The component directions *X* and *Y* refer to the element material coordinate system, which is explicitly defined for each element. The format of the Bulk Data entry for MAT2 is shown in Tab. A.2.

$$\begin{Bmatrix}\sigma_x\\ \sigma_x\\ \tau_{xy}\end{Bmatrix} = \begin{bmatrix}A_{11} & A_{12} & A_{13}\\ A_{12} & A_{22} & A_{23}\\ A_{13} & A_{23} & A_{33}\end{bmatrix}\left(\begin{Bmatrix}\varepsilon_x\\ \varepsilon_y\\ \gamma_{xy}\end{Bmatrix} - (T - T_{REF})\begin{Bmatrix}\alpha_x\\ \alpha_y\\ \alpha_{xy}\end{Bmatrix}\right), \tag{A.3}$$

where $\sigma_x, \sigma_y, \tau_{xy}$ – in-plane normal and shear stresses, $\varepsilon_x, \varepsilon_x, \gamma_{xy}$ – in-plane normal and shear strains.

$$\begin{Bmatrix}\tau_{xz}\\ \tau_{yz}\end{Bmatrix} = \begin{bmatrix}A_{11} & A_{12}\\ A_{12} & A_{22}\end{bmatrix}\begin{Bmatrix}\gamma_{xz}\\ \gamma_{yz}\end{Bmatrix}, \tag{A.4}$$

where τ_{xz}, τ_{yz} – interlaminar shear stresses, A_{ij} – interlaminar material property matrix, γ_{1z}, γ_{2z} – interlaminar shear strains.

Tab. A.2 MAT2 card

1	2	3	4	5	6	7	8	9	10
MAT2	MID	A_{11}	A_{12}	A_{13}	A_{22}	A_{23}	A_{33}	RHO	
	α_x	α_y	α_{xy}	T_{REF}	GE	S_t	S_c	S_s	
	MCSID								

The fields in Tab. A.2 have the next meaning:

S_t, S_c, S_s - stress limits for tension, compression, and shear are optionally supplied (these are used only to compute margins of safety in certain elements) and have no effect on the computational procedures.
MCSID - material coordinate system identification number.

Remarks:

- MCSID must be nonzero if PARAM, CURV is specified to extrapolate element centroid stresses or strains to grid points on plate and shell elements only. CQUAD4 element corner stresses are not supported by PARAM, CURV.
- Negative values for S_t, S_c, and S_s lead to no margins of safety being computed.
- If the MAT2 is referenced by the PCOMP/PCOMPG entry, the transverse shear flexibility for the referenced lamina is zero.

A.3 Layered Composite Element Property (PCOMP card) [1]

The PCOMP defines the properties of an n-ply composite material. The format of the Bulk Data entry for PCOMP is shown in Tab. A.3 below.

Tab. A.3 PCOMP card

1	2	3	4	5	6	7	8	9	10
PCOMP	PID	Z0	NSM	SB	FT	TREF	GE	LAM	
	MID1	T1	THETA1	SOUT1	MID2	T2	THETA2	SOUT2	
	MID3	T3	THETA3	SOUT3	-etc.-				

The fields in Tab. A.3 have the next meaning:

PID - Property identification number. (0 < Integer < 10000000).

Z0 - Distance from the reference plane to the bottom surface. (Real; Default = -0.5 times the element thickness.).

NSM - Nonstructural mass per unit area. (Real).

SB Allowable shear stress of the bonding material (allowable interlaminar shear stress). Required if FT is also specified. (Real > 0.0).

FT - Failure theory. The following theories are allowed (Character or blank. If blank, then no failure calculation will be performed):

- “HILL” for the Hill theory,
- “HOFF” for the Hoffman theory,
- “TSAI” for the Tsai-Wu theory,
- “STRN” for the Maximum Strain theory,

TREF - Reference temperature. (Real; Default = 0.0).

GE - Damping coefficient. (Real; Default = 0.0).

LAM - Laminate Options. (Character or blank, Default = blank):

- "Blank" All plies must be specified, and all stiffness terms are developed,
- "SYM" Only plies on one side of the element centerline are specified. The plies are numbered starting with 1 for the bottom layer. If an odd number of plies are desired, the center ply thickness (T1) should be half the actual thickness,
- "MEM" All plies must be specified, but only membrane terms (MID1 on the derived PSHELL entry) are computed,
- "BEND" All plies must be specified, but only bending terms (MID2 on the derived PSHELL entry) are computed,
- "SMEAR" All plies must be specified, stacking sequence is ignored MID1=MID2 on the derived PSHELL entry and MID3, MID4 and TS/T and $12I/T^3$ terms are set as blanks),
- "SMCORE" All plies must be specified, with the last ply specifying core properties and the previous plies specifying face sheet properties. The stiffness matrix is computed by placing half the face sheet thicknesses above the core and the other half below with the result that the laminate is symmetric about the mid-plane of the core. Stacking sequence is ignored in calculating the face sheet stiffness.

MIDi - Material ID of the various plies. The plies are identified by serially numbering them from 1 at the bottom layer. The MIDs must refer to MAT2, MAT8, etc. Bulk Data entries. (Integer > 0 or blank, except MID1 must be specified.)

Ti - Thicknesses of the various plies. (Real or blank, except T1 must be specified.)

THETAi - Orientation angle of the longitudinal direction of each ply with the material axis of the element. (If the material angle on the element connection entry is 0.0, the material axis and side 1-2 of the element coincide.) The plies are to be numbered serially starting with 1 at the bottom layer. The bottom layer is defined as the surface with the largest -Z value in the element coordinate system. (Real; Default = 0.0).

SOUTi - Stress or strain output request. (Character: "YES" or "NO"; Default = "NO").

A.4 Shell Element Property (PSHELL card) [1]

PSHELL defines the membrane, bending, transverse shear, and coupling properties of thin shell elements. The format of the Bulk Data entry for PCOMP is shown in Tab. A.4 below.

Tab. A.4 PSHELL card

1	2	3	4	5	6	7	8	9	10
PSHELL	PID	MID1	T	MID2	$12I/T^3$	MID3	TS/T	NSM	
Z1	Z2	MID4							

The fields in Tab. A.4 have the next meaning:

PID - Property identification number. (Integer > 0).

MID1 - Material identification number for the membrane. (Integer > 0 or blank).

T - Default membrane thickness for Ti on the connection entry. If T is blank then the thickness must be specified for Ti on the CQUAD4, CTRIA3, CQUAD8, and CTRIA6 entries. (Real or blank) Average thickness if TFLAG = 1.

MID2 - Material identification number for bending. (Integer > -1 or blank)

12I/T^3 - Bending moment of inertia ratio, $12I/T^3$. Ratio of the actual bending moment inertia of the shell, I, to the bending moment of inertia of a homogeneous shell, $T^3/12$. The default value is for a homogeneous shell. (Real > 0.0; Default = 1.0).

MID3 - Material identification number for transverse shear. If MID2 is blank or -1, then

MID3 must be blank. (Integer > 0 or blank).

TS/T - Transverse shear thickness ratio, T_s/T. Ratio of the shear thickness, T_s, to the membrane thickness of the shell, T. The default value is for a homogeneous shell. (Real > 0.0; Default = 0.833333).

NSM - Nonstructural mass per unit area. (Real).

Z1, Z2 - Fiber distances for stress calculations. The positive direction is determined by the right-hand rule, and the order in which the grid points are listed on the connection entry. See Remark 11. for defaults. (Real or blank).

MID4 - Material identification number for membrane-bending coupling. See Remarks. (Integer > 0 or blank, must be blank unless MID1 > 0 and MID2 > 0, may not equal MID1 or MID2.)

Remarks:

The results of leaving a MIDi field blank are:

- MID1 - no membrane or coupling stiffness,
- MID2 - no bending, coupling, or transverse shear stiffness,
- MID3 - no transverse shear flexibility,
- MID4 - no bending-membrane coupling unless ZOFFS is specified on the connection entry.

A.5 "Global_model.bdf" – BDF file for the global model

```
$ MSC.Nastran input file created on August    08, 2021 at 16:42:22
by
$ Patran 2021
$ Direct Text Input for Nastran System Cell Section
$ Direct Text Input for File Management Section
$ Direct Text Input for Executive Control
```

```
$ Linear Static Analysis, Database
SOL 101
CEND
$ Direct Text Input for Global Case Control Data
TITLE = MSC.Nastran job created on 19-Jul-21 at 13:02:49
ECHO = NONE
SUBCASE 1
   SUBTITLE=Default
   SPC = 2
   LOAD = 2
   DISPLACEMENT(SORT1,REAL)=ALL
   SPCFORCES(SORT1,REAL)=ALL
   STRAIN(SORT1,REAL,VONMISES,STRCUR,BILIN)=ALL
   STRESS(SORT1,REAL,VONMISES,BILIN)=ALL
$ Direct Text Input for this Subcase
LINE=99999
BEGIN BULK
INCLUDE 'MAT2_1.bdf'
INCLUDE 'MAT2_2.bdf'
INCLUDE 'MAT2_3.bdf'
INCLUDE 'MAT2_4.bdf'
$ Direct Text Input for Bulk Data
PARAM    POST    0
PARAM   PRTMAXIM YES
$ Elements and Element Properties for region : Panel_1
PSHELL   1       1      &t1[8;(F8.6)]& 1
$ Pset: "Panel_1" will be imported as: "pshell.1"
CQUAD4   130     1       121     122     438     128     90.
                                 …
                                 …
                                 …
CQUAD4   162     1       455     159     166     165     90.
$ Elements and Element Properties for region : Panel_2
PSHELL   2       2      &t2[8;(F8.6)]& 2
$ Pset: "Panel_2" will be imported as: "pshell.2"
CQUAD4   127     2       93      119     436     95      90.
                                 …
                                 …
                                 …
CQUAD4   159     2       453     156     163     162     90.
$ Elements and Element Properties for region : Panel_3
PSHELL   3       3      &t3[8;(F8.6)]& 3
$ Pset: "Panel_3" will be imported as: "pshell.3"
CQUAD4   166     3       163     164     458     170     90.
                                 …
                                 …
                                 …
CQUAD4   198     3       475     201     208     207     90.
$ Elements and Element Properties for region : Panel_4
PSHELL   4       4      &t4[8;(F8.6)]& 4
$ Pset: "Panel_4" will be imported as: "pshell.4"
CQUAD4   163     4       105     161     456     107     90.
                                 …
                                 …
```

```
                                …
CQUAD4   195     4       473     198     205     204     90.
$ Elements and Element Properties for region : Shell_45_X
$ Composite Property Reference Material: Laminate_45
$ Composite Material Description :
PCOMP    5
*        5               1.2700001-4     45.             YES
*        5               1.2700001-4     -45.            YES
                                …
                                …
                                …
*        5               1.2700001-4     45.             YES
$ Pset: "Shell_45_X" will be imported as: "pcomp.5"
CQUAD4   73      5       1       2       119     93      0.
                                …
                                …
                                …
CQUAD4   126     5       341     342     166     165     0.
$ Elements and Element Properties for region : Shell_45_Z
$ Composite Property Reference Material: Laminate_45
$ Composite Material Description :
PCOMP    6
*        5               1.2700001-4     45.             YES
                                …
                                …
                                …
*        5               1.2700001-4     45.             YES
$ Pset: "Shell_45_Z" will be imported as: "pcomp.6"
CQUAD4   1       6       1       2       302     301     90.
                                …
                                …
                                …
CQUAD4   114     6       374     88      205     198     0.
$ Referenced Material Records
$ Material Record : T300-N5208
$ Description of Material : Date: 19-Jul-21           Time: 11:21:35
MAT8*    5               1.27553+11      1.303109+10     .3
*        6.4121236+9                                     1577.75
$ Nodes of the Entire Model
GRID     1               0.      0.      0.
GRID*    2                               .373383313417435 0.
*        0.
                                …
                                …
                                …
GRID*    475                            1.86691653728485 .381
*       3.24802446365356
$ Loads for Load Case : Default
SPCADD   2       1
LOAD     2       1.      1.      1       1.      3       1.      4
$ Displacement Constraints of Load Set : Fixed_Root
SPC1     1       123456  1       THRU    7
SPC1     1       123456  93      119     120     121     122     123
```

```
         124
$ Nodal Forces of Load Set : F3
FORCE    4       85      0      &F3[8;(F8.2)]& 0.      1.      0.
$ Nodal Forces of Load Set : F2
FORCE    3       88      0      &F2[8;(F8.2)]& 0.      1.      0.
$ Nodal Forces of Load Set : F1
FORCE    1       91      0      &F1[8;(F8.2)]& 0.      1.      0.
$ Referenced Coordinate Frames
ENDDATA 59c46d0d
```

A.6 "Global_model_2.bdf" – BDF file of the detailed global model

```
$ MSC.Nastran input file created on August    20, 2021 at 10:27:57
by
$ Patran 2021
$ Direct Text Input for Nastran System Cell Section
$ Direct Text Input for File Management Section
$ Direct Text Input for Executive Control
$ Buckling Analysis, Database
SOL 105
CEND
$ Direct Text Input for Global Case Control Data
TITLE = MSC.Nastran job created on X19-Jul-21 at 13:02:49
ECHO = NONE
SUBCASE 1
   SUBTITLE=Default
   SPC = 2
   LOAD = 2
   DISPLACEMENT(SORT1,REAL)=ALL
   SPCFORCES(SORT1,REAL)=ALL
   STRAIN(SORT1,REAL,VONMISES,STRCUR,BILIN)=ALL
SUBCASE 2
   SUBTITLE=Default
   SPC = 2
   METHOD = 1
   VECTOR(SORT1,REAL)=ALL
   SPCFORCES(SORT1,REAL)=ALL
   STATSUB = 1
$ Direct Text Input for this Subcase
LINE=99999
BEGIN BULK
INCLUDE 'PCOMP_1.bdf'
INCLUDE 'PCOMP_2.bdf'
INCLUDE 'PCOMP_3.bdf'
INCLUDE 'PCOMP_4.bdf'
$ Direct Text Input for Bulk Data
PARAM    POST    0
PARAM   PRTMAXIM YES
EIGRL    1                       20      0
$ Elements and Element Properties for region : Panel_1
$ Pset: "Panel_1" will be imported as: "pcomp.1"
CQUAD4   130     1       121     122     438     128     90.
                                 ...
                                 ...
                                 ...
```

```
CQUAD4   162     1       455     159     166     165     90.
$ Elements and Element Properties for region : Panel_2
$ Pset: "Panel_2" will be imported as: "pcomp.2"
CQUAD4   127     2       93      119     436     95      90.
                                 …
                                 …
                                 …
CQUAD4   159     2       453     156     163     162     90.
$ Elements and Element Properties for region : Panel_3
$ Pset: "Panel_3" will be imported as: "pcomp.3"
CQUAD4   166     3       163     164     458     170     90.
                                 …
                                 …
                                 …
CQUAD4   198     3       475     201     208     207     90.
$ Elements and Element Properties for region : Panel_4
$ Pset: "Panel_4" will be imported as: "pcomp.4"
CQUAD4   163     4       105     161     456     107     90.
                                 …
                                 …
                                 …
CQUAD4   195     4       473     198     205     204     90.
$ Elements and Element Properties for region : Shell_Z
$ Composite Property Reference Material: Laminate_45
$ Composite Material Description :
PCOMP    5
*        1               1.2700001-4     45.             YES
                                 …
                                 …
                                 …
*        1               1.2700001-4     45.             YES
$ Pset: "Shell_Z" will be imported as: "pcomp.5"
CQUAD4   1       5       1       2       302     301     90.
                                 …
                                 …
                                 …
CQUAD4   114     5       374     88      205     198     0.
$ Elements and Element Properties for region : Shell_X
$ Composite Property Reference Material: Laminate_45
$ Composite Material Description :
PCOMP    6
*        1               1.2700001-4     45.             YES
                                 …
                                 …
                                 …
*        1               1.2700001-4     45.             YES
$ Pset: "Shell_X" will be imported as: "pcomp.6"
CQUAD4   73      6       1       2       119     93      0.
                                 …
                                 …
                                 …
CQUAD4   126     6       341     342     166     165     0.
$ Referenced Material Records
```

```
$ Material Record : T300-N5208
$ Description of Material : Date: 19-Jul-21           Time: 11:21:35
MAT8*    1               1.27553+11      1.303109+10     .3
*        6.4121236+9                                     1577.75
$ Nodes of the Entire Model
GRID     1               0.      0.      0.
GRID*    2                               .373383313417435 0.
*        0.
                                  …
                                  …
                                  …
GRID*    475                             1.86691653728485 .381
*       3.24802446365356
$ Loads for Load Case : Default
SPCADD   2       1
LOAD     2       1.      1.      1       1.      3       1.      4
$ Displacement Constraints of Load Set : Fixed_Root
SPC1     1       123456  1       THRU    7
SPC1     1       123456  93      119     120     121     122     123
         124
$ Nodal Forces of Load Set : F3
FORCE    4       85      0       1877.15 0.      1.      0.
$ Nodal Forces of Load Set : F2
FORCE    3       88      0       1877.15 0.      1.      0.
$ Nodal Forces of Load Set : F1
FORCE    1       91      0       3798.78 0.      1.      0.
$ Referenced Coordinate Frames
ENDDATA acf957de
```

A.7 "Panel_Local.bdf" – BDF file for the local FE model of a panel

```
$ MSC.Nastran input file created on August    14, 2021 at 01:00:49
by
$ Patran 2021
$ Direct Text Input for Nastran System Cell Section
$ Direct Text Input for File Management Section
$ Direct Text Input for Executive Control
$ Buckling Analysis, Database
SOL 105
CEND
$ Direct Text Input for Global Case Control Data
TITLE = MSC.Nastran job created on X19-Jul-21 at 13:02:49
ECHO = NONE
SUBCASE 1
   SUBTITLE=Default
   SPC = 2
   LOAD = 1
   DISPLACEMENT(SORT1,REAL)=ALL
   SPCFORCES(SORT1,REAL)=ALL
   STRAIN(SORT1,REAL,VONMISES,STRCUR,BILIN)=ALL
SUBCASE 2
   SUBTITLE=Default
   SPC = 2
   METHOD = 1
   VECTOR(SORT1,REAL)=ALL
```

```
   SPCFORCES(SORT1,REAL)=ALL
   STATSUB = 1
$ Direct Text Input for this Subcase
LINE=99999
BEGIN BULK
INCLUDE 'PCOMP.bdf'
INCLUDE 'DISPALCEMENTS.bdf'
$ Direct Text Input for Bulk Data
PARAM    POST    0
PARAM   PRTMAXIM YES
EIGRL    1       0.      100.    2       0
$ Elements and Element Properties for region : Laminate_Shell
$ Pset: "Laminate_Shell" will be imported as: "pcomp.1"
CQUAD4   1       1       16      326     339     338     90.
                                 …
                                 …
                                 …
CQUAD4   288     1       635     636     7       648     90.
$ Referenced Material Records
$ Material Record : T300-N5208
$ Description of Material : Date: 19-Jul-21           Time: 11:21:35
MAT8*    1               1.27553+11      1.303109+10     .3
*        6.4121236+9                                     1577.75
$ Multipoint Constraints of the Entire Model
RSPLINE  289             16      338     123456  351     123456  364
         123456  15
RSPLINE  290             15      390     123456  403     123456  416
         123456  14
RSPLINE  291             14      442     123456  455     123456  468
         123456  13
RSPLINE  292             13      494     123456  507     123456  520
         123456  12
RSPLINE  293             12      546     123456  559     123456  572
         123456  11
RSPLINE  294             11      598     123456  611     123456  624
         123456  10
RSPLINE  295             10      638     123456  639     123456  640
         123456  9
RSPLINE  296             9       642     123456  643     123456  644
         123456  8
RSPLINE  297             8       646     123456  647     123456  648
         123456  7
RSPLINE  298             7       636     123456  623     123456  610
         123456  6
RSPLINE  299             6       584     123456  571     123456  558
         123456  5
RSPLINE  300             5       532     123456  519     123456  506
         123456  4
RSPLINE  301             4       480     123456  467     123456  454
         123456  3
RSPLINE  302             3       428     123456  415     123456  402
         123456  2
RSPLINE  303             2       376     123456  363     123456  350
         123456  1
RSPLINE  304             1       336     123456  335     123456  334
```

```
         123456  18
RSPLINE  305             18      332     123456  331     123456  330
         123456  17
RSPLINE  306             17      328     123456  327     123456  326
         123456  16
$ Nodes of the Entire Model
GRID*    1                               1.12014997005463 0.
*        0.
GRID*    2                               1.12014997005463 0.
*       .295275002717972
                                  …
                                  …
                                  …
GRID*    648                             1.02680420875549 0.
*       1.77164995670319
$ Loads for Load Case : Default
SPCADD   2       1
$ Enforced Displacements for Load Set : Displacements
$ Displacement Constraints of Load Set : Displacements
SPC1     1       123456  1       THRU    18
$ Referenced Coordinate Frames
ENDDATA db936f2f
```

Appendix B Developed python scripts

B.1 "MAT2_writer.py" - Python code for generation of BDF files with MAT2 card

```
import argparse

def write_MAT2(A_file, MAT2_file, Panel_Num, ADensity):

    A = []
    lines = []
    pcomp_s = []

    with open(A_file, 'r') as fr:
        A = [line.split() for line in fr]
    fr.close()

    Fail = 1.0

    if Fail > 0.0:
        for i in range(0, len(A)):
            if float(A[0][i]) <= 0:
                Fail = 0.0


# If Fail < 1 the upper-level analysis is failed, since some of Aij <= 0
if Fail > 0.0:
    MAT2 = [str(Panel_Num), str('{:11.9e}'.format(float(A[0][0]))),
str('{:11.9e}'.format(float(A[0][1]))), str(0.0), "\n*       ",
str('{:11.9e}'.format(float(A[0][2]))), str(0.0),
str('{:11.9e}'.format(float(A[0][3]))),
str('{:11.9e}'.format(float(ADensity)))]
# Strings are being written to the file

        for i in range(0, len(MAT2)):
            if i != 4 and len(MAT2[i]) < 16:
                for j in range(0, 16 - len(MAT2[i])):
                    MAT2[i] = MAT2[i] + " "

        fw = open(MAT2_file, 'w')
        fw.write("$ Material Record: Panel_" + str(Panel_Num) + "\n" + "$
Description of Material:" + "$ Membrane material properties:" + "\n" +
"MAT2*   ")
        for i in range(0, len(MAT2)):
            fw.write(MAT2[i])
        fw.write("\n")
        fw.close()
    return


def createparser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-A_file')
    parser.add_argument('-MAT2_file')
    parser.add_argument('-PanelNum')
    parser.add_argument('-Density')
    return parser
if __name__ == '__main__':
     parser = createparser()
     namespace = parser.parse_args()
```

```
write_MAT2(namespace.A_file, namespace.MAT2_file, namespace.PanelNum,
namespace.Density)
```

B.2 "PCOMP_global_writer.py" - Python code for generation of BDF files with PCOMP card for the detailed global FE model

```
import argparse, os, time
from datetime import datetime

# Class defining the structure of data required for a guide
class TGuide:

  def __init__(self, AN_of_Panels, AL):

    self.Angles = FillGuideAngles(AL) # Array of the stacking sequence
    self.n = [] # array containing number of layers for each panel
    self.RF = [] # array containing RFmin for each panel
    self.F = [] # array containing fitness for each panel
    for i in range(0, AN_of_Panels):
      self.n.append(-1)
      self.RF.append(-1)
      self.F.append(-1)
    self.Fitness = -1

# Function reading guides from Best_Guides.txt file
def ReadGuides(filer, N_of_Panels):

  GUIDES = []
  FRContent = []

  counter = 0
  while not os.path.isfile(filer):
    time.sleep(0.01)
    if counter == 0:
      counter = 1

  with open(filer, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  for i in range(0, len(FRContent)):
    AGuide = TGuide(N_of_Panels, len(FRContent))
    AGuide.Angles = []

    fail = 0
    # Read n-vector from List_of_Guides file
    for j in range(0, N_of_Panels):
      AGuide.n[j] = int(float(FRContent[i][j]))

    k = 0
    # Read RF-vector from List_of_Guides file
    for j in range(N_of_Panels, 2 * N_of_Panels):
      AGuide.RF[k] = float(FRContent[i][j])
      k += 1

    k = 0
    # Read F-vector from List_of_Guides file
    for j in range(2 * N_of_Panels, 3 * N_of_Panels):
      AGuide.F[k] = float(FRContent[i][j])
      k += 1
```

```
    # Read Angles-vector from List_of_Guides file
    for j in range(3 * N_of_Panels + 1, len(FRContent[i])):
      AGuide.Angles.append(float(FRContent[i][j]))

    # Read fitness values from List_of_Guides file
    AGuide.Fitness = float(FRContent[i][3 * N_of_Panels])
    GUIDES.append(AGuide)

  return GUIDES

# Function for writing PCOMP card to a .bdf file
def write_pcomp(to, AN_of_Panels, index):

    FRContent = []
    lines = []
    pcomp_s = []
    GUIDES = []

    index = index - 1
    # Read guides from Best_Guides.txt file
    GUIDES = ReadGuides("Best_Guides.txt", AN_of_Panels)

    # Create strings of the *.bdf files for each panel
    for i in range(len(GUIDES[index].n)):
      k = 0
      for j in range(1, 2 * GUIDES[index].n[i] + 1):
        if j <= GUIDES[index].n[i]:
          k = len(GUIDES[index].Angles) - GUIDES[index].n[i] + j - 1
        if j > GUIDES[index].n[i]:
          k = len(GUIDES[index].Angles) + GUIDES[index].n[i] - j
        if GUIDES[index].Angles[k] < -9:
          sp = '    '
        if GUIDES[index].Angles[k] > 9 or -9 <= GUIDES[index].Angles[k] <
0:
          sp = '    '
        if GUIDES[index].Angles[k] >= 0 and GUIDES[index].Angles[k] <= 9:
          sp = '     '
        lines.append('     1      ' + str('{:6.5f}'.format(to).strip('0'))
+ '   ' + str('{:3.1f}'.format(GUIDES[index].Angles[k])) + sp + ' YES')

    # Create *.bdf files for each panel
      fw = open("PCOMP_" + str(i + 1) + ".bdf" , 'w')
      fw.write("$ Composite Material Description :" + "\n" + "PCOMP    " +
str(i + 1) + "\n")
      fw.close()

      s_pcomp = '    '
      j = 0

    # Write lines into *.bdf files
      for k in range(len(lines)):
        j += 1
        if j <= 2:
          s_pcomp = s_pcomp + lines[k]
          if j == 2 or k == len(lines) - 1:
            fw = open("PCOMP_" + str(i + 1) + ".bdf", 'a')
            fw.write(s_pcomp + '\n')
            fw.close()
            j = 0
            s_pcomp = '    '
```

```
      lines = []
    return

# Function for filling in the array of stacking sequence of a guide
def FillGuideAngles(L):

  AAngles = []

  for k in range(0, L):
    AAngles.append(0)

  return AAngles


def createparser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-to')
    parser.add_argument('-AN_of_Panels')
    parser.add_argument('-index')
    return parser


if __name__ == '__main__':
     parser = createparser()
     namespace = parser.parse_args()

write_pcomp(float(namespace.to), int(float(namespace.AN_of_Panels)),
int(float(namespace.index)))
```

B.3 "Genetics_Guides_Opt.py" – Python code for the main GA-level optimization algorithm

```
import time
from datetime import datetime
import os
from shutil import copyfile
from math import fabs

# Class defining the structure of data required for a guide
class TGuide:

  def __init__(self, AN_of_Panels):

    self.Angles = [] # array of the stacking sequence
    self.n = [] # array containing number of layers for each panel
    self.RF = [] # array containing RFmin for each panel
    self.F = [] # array containing fitness for each panel
    for i in range(0, AN_of_Panels):
      self.n.append(-1)
      self.RF.append(-1)
      self.F.append(-1)
    self.Fitness = -1

# Class defining the Options structure of the algorithm
class TOptions:

  def __init__(self):
    self.L = 0
    self.N = 0
    self.StopCriterion = ""
    self.Eps = 0.0
```

```
    self.BestPool = 1
    self.Ksi = 0.0
    self.GlobalPath = ""


# Class defining the Input structure of the algorithm
class TInput:

  def __init__(self):
    self.MaxGen = 5
    self.Fail = 0.0
    self.N_of_Panels = 1.0


# Function reading the Options of the algorithm defined by a user
def ReadOptions():

  global Options

  with open("Genetics_Guides_Opt.options", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Options.L = int(FRContent[0][1])
  Options.N = int(FRContent[1][1])
  Options.StopCriterion = str(FRContent[2][1])
  Options.Eps = float(FRContent[3][1])
  Options.BestPool = int(FRContent[4][1])
  Options.Ksi = float(FRContent[5][1])
  Options.GlobalPath =
"d:\!Ph.D.course\!Work_progress\Optimus_projects\Global_level-r.3.1\\"

  return Options

# Function reading the Input parameters of the algorithm defined by the
Optimus from the PROBLEM file
def ReadInput():

  global Input
  with open("PROBLEM", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Input.MaxGen = int(FRContent[2][4])
  Input.Fail = int(FRContent[3][4])
  Input.N_of_Panels = int(float(FRContent[4][5]))
  return Input

# Initialysing the structured constant containing the Options of the
algorithm
Options = TOptions()
ReadOptions()

# Initialyzing the structured constant containing the Input for the
algorithm
Input = TInput()
ReadInput()

# Main Optimization function of the algorithm
def Optimize():
```

```
  global Options
  global Input

  GUIDES = []
# Fial parameter shows if the upper-level analysis has failed
  if Input.Fail != 1.0:

    I = 0
    stop = 0
    # If Fial = 0 optimization is cancelled
    while I <= Input.MaxGen and stop == 0:
      # Write data for the next step of Optimus analysis
      WriteNEWPOINTS(I)
      # Read results of Optimus analysis
      ReadResults()
      # For generations > 0 do:
      if I > 0:
        # Read guides from the List of Guides.txt file
        GUIDES = ReadGuides("List_of_Guides.txt", Input.N_of_Panels,
"Ordinary")
        # Check stop criteria for guides
        stop = Stoper(GUIDES)
      I += 1

    # Delete duplicates in the Best_Guides.txt file
    Del_Duplicates("Best_Guides.txt")
    # Read guides from the Best Guides.txt file
    GUIDES = ReadGuides("Best_Guides.txt", Input.N_of_Panels, "")
    if len(GUIDES) > 1:
      # Sort best guides in ascending order of their fitness
      GUIDES = SortGuides(GUIDES)
      # Write sorted guides into the Best guides.txt file
      Write_Guides(GUIDES, "Best_Guides.txt", Input.N_of_Panels)
"Best_Guides.txt\n\n")
    # Copy the Best_guides.txt file to the upper level
    copyfile("Best_Guides.txt", Options.GlobalPath + "Best_Guides.txt")
GlobalPath\n\n")

    if os.path.isfile("List_of_Guides.txt"):
      os.remove("List_of_Guides.txt")
  return


def CleanOptimusDir():
  open("WAITFOROPTIMUS","w").close()
  os.remove("Genetics_Guides_Opt.running")

# Function for writing data for the next step of Optimus analysis
def WriteNEWPOINTS(Gen):

  global Input
  global EXP
  global dir_path

  fw = open("NEWPOINTS", 'w')

  fw.write("CONTINUE" + "\n")
  fw.write(str(Gen) + "\n")
  fw.close()

  open("WAITFOROPTIMUS", "w").close()
```

```
# Function reading results of Optimus analysis from the SUMMARY file
def ReadResults():

  FRContent = []

  i = 0
  while os.path.isfile("WAITFOROPTIMUS"):
    if not os.path.isfile("TERMINATE") and os.path.isfile("LOCK.OPTIMUS"):
      i += 1
      if i == 1:
        time_start = datetime.now()
      time.sleep(0.01)
    else:
      CleanOptimusDir()
      break


  with open("SUMMARY", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

# Function reading guides from a file with guides
def ReadGuides(filer, AN_of_Panels, mode):

  global Options

  GUIDES = []
  FRContent = []

  with open(filer, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  if mode == "Ordinary":
    i0 = 1
  else:
    i0 = 0

  for i in range(i0, len(FRContent)):
    AGuide = TGuide(AN_of_Panels)
    # Read n-vector from List_of_Guides file
    for j in range(0, AN_of_Panels):
      AGuide.n[j] = int(float(FRContent[i][j]))
    # Read RF-vector from List_of_Guides file
    for j in range(AN_of_Panels, 2 * AN_of_Panels):
      AGuide.RF[j - AN_of_Panels] = float(FRContent[i][j])
    # Read F-vector from List_of_Guides file
    for j in range(2 * AN_of_Panels, 3 * AN_of_Panels):
      AGuide.F[j - 2 * AN_of_Panels] = float(FRContent[i][j])
    AGuide.Fitness = float(FRContent[i][3 * AN_of_Panels])
    # Read Angles-vector from List_of_Guides file
    for j in range(3 * AN_of_Panels + 1, len(FRContent[i])):
      AGuide.Angles.append(float(FRContent[i][j]))
    GUIDES.append(AGuide)
  return GUIDES


# Function writing guides from a file with guides
```

```
def Write_Guides(AGUIDES, file, AN_of_Panels):

    global Options

    sp_n = []
    sp_RF = []
    sp_F = []

    fw = open(file, 'w')

    for I in range(0, len(AGUIDES)):
      for J in range(0, AN_of_Panels):
        if AGUIDES[I].n[J] < 0:
          sp_n.append(" ")
        else:
          sp_n.append("  ")

        if AGUIDES[I].RF[J] < 0:
          sp_RF.append(" ")
        else:
          sp_RF.append("  ")

        if AGUIDES[I].F[J] < 0:
          sp_F.append(" ")
        else:
          sp_F.append("  ")

      if AGUIDES[I].Fitness < 0:
        sp_f = " "
      else:
        sp_f = "  "

    for I in range(0, len(AGUIDES)):
      for J in range(0, AN_of_Panels):
        if J == 0:
          fw.write(str(AGUIDES[I].n[J]))
        else:
          fw.write(sp_n[J] + str(AGUIDES[I].n[J]))

      for J in range(0, AN_of_Panels):
        fw.write(sp_RF[J] + str('{:.2e}'.format(AGUIDES[I].RF[J])))

      for J in range(0, AN_of_Panels):
        fw.write(sp_F[J] + str('{:.3e}'.format(AGUIDES[I].F[J])))

      fw.write(sp_f + str('{:.3e}'.format(AGUIDES[I].Fitness)) + "    ")

      for J in range(0, Options.L):

        if AGUIDES[I].Angles[J] < 0:
          sp1 = " "
        else:
          sp1 = "  "

        if AGUIDES[I].Angles[J] >= -9.9 and AGUIDES[I].Angles[J] <= 9.9:
          sp2 = "  "
        else:
          sp2 = " "

        fw.write(sp1 + str('{:3.1f}'.format(AGUIDES[I].Angles[J])) + sp2)
      fw.write("\n")
```

```
    fw.close()
    return


# Function checking stop criteria for guides
def Stoper(AGUIDES):

  global Options
  BestInds = []

  stop = 0

  if Options.StopCriterion == "FitnessTolerance" or Options.StopCriterion
== "FitnessToleranceBestGuidesScatter":
    for i in range(0, Options.N):
      if AGUIDES[i].Fitness <= Options.Eps/100:
        stop = 1
        break

  if Options.StopCriterion == "BestGuidesScatter" or Options.StopCriterion
== "FitnessToleranceBestGuidesScatter":
    if stop == 0:
      Temp = AGUIDES[0]
      for i in range(0, Options.N):
        for j in range(0, Options.N):
          if AGUIDES[i].Fitness < AGUIDES[j].Fitness:
            Temp = AGUIDES[i]
            AGUIDES[i] = AGUIDES[j]
            AGUIDES[j] = Temp

      BestInds = AGUIDES[0 : Options.BestPool]

      FitnessAve = 0

      for i in range(0, Options.BestPool):
        FitnessAve = FitnessAve + BestInds[i].Fitness

      FitnessAve = FitnessAve/Options.BestPool

      stop = 1
      for i in range(0, Options.BestPool):
        if BestInds[i].Fitness/FitnessAve > Options.Ksi/100 + 1:
          stop = 0
  return stop


# Function deleting duplicate guides from a file
def Del_Duplicates(filer):

  f = open(filer, 'r')
  lstResul = f.readlines()
  f.close()
  if len(lstResul) > 1:
    datos = []
    for lstRspn in lstResul:
      datos.append(lstRspn)
    lstSize = len(datos)
    i = 0
    f = open("WODublicates.txt", 'w')
    while i < lstSize:
```

```
      if i == 0:
        f.writelines(datos[i])
      else:
        if (str(datos[i - 1].strip())).replace(' ', '') !=
(str(datos[i].strip())).replace(' ', ''):
          f.writelines(datos[i])
      i = i + 1
    f.close()
    copyfile("WODublicates.txt", filer)
  return


# Function sorting guides in a file
def SortGuides(AGUIDES):

  Temp = AGUIDES[0]
  for i in range(0, len(AGUIDES)):
    for j in range(0, len(AGUIDES)):
      if AGUIDES[i].Fitness < AGUIDES[j].Fitness:
        Temp = AGUIDES[i]
        AGUIDES[i] = AGUIDES[j]
        AGUIDES[j] = Temp
      else:
        if AGUIDES[i].Fitness == AGUIDES[j].Fitness:
          FSum_i = 0
          FSum_j = 0
          for k in range(0, len(AGUIDES[i].F)):
            FSum_i = FSum_i + AGUIDES[i].F[k]
          for k in range(0, len(AGUIDES[j].F)):
            FSum_j = FSum_j + AGUIDES[j].F[k]
          if FSum_i < FSum_j:
            Temp = AGUIDES[i]
            AGUIDES[i] = AGUIDES[j]
            AGUIDES[j] = Temp
  return AGUIDES


if __name__ == '__main__':
  if not os.path.isfile("Genetics_Guides_Opt.running"):
    open("Genetics_Guides_Opt.running","w").close()

  Optimize()
  NP = open("NEWPOINTS", 'w')
  NP.write("STOP\n")
  NP.close()
  CleanOptimusDir()
```

B.4 “Guides_Operator.py” - Python code for GA operations

```
from __future__ import print_function

import os
import random
import time
from datetime import datetime
import argparse
from shutil import copyfile

Stiffness = []

angle_min = 0.0
```

```
angle_max = 90.0
angle_step = 15.0

# Class defining the Options structure of the algorithm
class TOptions:

  def __init__(self):
    self.L = 0
    self.N = 0
    self.LocalPath = ""
    self.GuidePath = ""


# A structured constant containing the Options of the algorithm
Options = TOptions()

# Array with angles allowed for stacking sequence defined by a user
angles = [angle_min]

while angles[len(angles) - 1] < angle_max:
  angles.append(angles[len(angles) - 1] + angle_step)

# Class defining the structure of data required for a guide
class TGuide:

  def __init__(self, AN_of_Panels, mode):

    self.Angles = FillGuideAngles(mode) # array of the stacking sequence
    self.n = [] # array containing number of layers for each panel
    self.RF = [] # array containing RFmin for each panel
    self.F = [] # array containing fitness for each panel
    for i in range(0, AN_of_Panels):
      self.n.append(-1)
      self.RF.append(-1)
      self.F.append(-1)
    self.Fitness = -1


# Class defining guides as parents to mate
class TParents:

  def __init__(self, AN_of_Panels):
    self.A = TGuide(AN_of_Panels, "")
    self.M = TGuide(AN_of_Panels, "")

# A structured constant containing the Options of the algorithm
Options = TOptions()

# The list containing guides
Generation = []

# Function reading the Options of the algorithm defined by a user
def ReadOptions():

  global Options

  with open("Genetics_Guides_Opt.options", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Options.L = int(FRContent[0][1])
```

```
  Options.N = int(FRContent[1][1])
  Options.LocalPath = str(FRContent[6][1])
  Options.GuidePath = str(FRContent[7][1])
  return Options


ReadOptions()


# Main function for doing operations with guides
def Operate_with_Guides(AEXP, AGen, AN_of_Panels, AMode):

     GUIDES = []
     global Options

     if AGen == 0:
         # For the first generation generate guides randomly of read them
from a file
         if AMode == "GenerateGuides":
           GUIDES = Generate_Guides(AN_of_Panels)
         else:
           GUIDES = ReadGuides(Options.GuidePath + "List_of_Guides.txt",
AN_of_Panels)
         filew = "List_of_Guides.txt"
     else:
         # For the next generations do
         # Copy List of Guides.txt from the GuidePath directory to the
current directory
         copyfile(Options.GuidePath + "List_of_Guides.txt",
"List_of_Guides.txt")
         # Perform genetic operations
         GUIDES = Mate_Guides("List_of_Guides.txt", AN_of_Panels)
         filew = "Parents.txt"

     # Write guides the Parents.txt(or List_of_Guides.txt) file
     Write_Guides(GUIDES, filew, AN_of_Panels)

     fw = open("N_of_Guides.txt", 'w')
     fw.write(str(Options.N))
     fw.close()

     # Copy List_of_Guides.txt/Parents.txt from the current directory to
the GuidePath directory
     copyfile(filew, Options.GuidePath + filew)
     # Copy List of Guides.txt/Parents.txt from the current directory to
the LocalPath directory
     copyfile(filew, Options.LocalPath + filew)


# Function generating guides randomly
def Generate_Guides(AN_of_Panels):

    global Options

    # Fill in the 1st generation of Guides randomly
    AGUIDES = []

    for I in range(0, Options.N):
        AGuide = TGuide(AN_of_Panels, "1st_Gen")
        AGUIDES.append(AGuide)
    return AGUIDES
```

```
# Function calling genetic operations for parent-guides
def Mate_Guides(filer, AN_of_Panels):

    global Options
    Generation = []
    # Read guides from the List_of_Guides.txt file
    Generation = ReadGuides(filer, AN_of_Panels)
    # Select guides for crossing over
    AParents = Selection(Generation, AN_of_Panels)
    # Perform crossing over of selected guides
    AParents = Crossover(AParents, AN_of_Panels)
    # Perform mutation of children-guides
    AParents = Mutation(AParents)
    AGUIDES = [AParents.A, AParents.M]
    return AGUIDES


# Function reading guides from a file with guides
def ReadGuides(filer, AN_of_Panels):
  global Options

  GUIDES = []
  FRContent = []

  with open(filer, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  for i in range(1, len(FRContent)):
    AGuide = TGuide(AN_of_Panels, "")
    AGuide.Angles = []
    # Read n-vector from List_of_Guides file
    for j in range(0, AN_of_Panels):
      AGuide.n[j] = int(float(FRContent[i][j]))
    # Read RF-vector from List_of_Guides file
    for j in range(AN_of_Panels, 2 * AN_of_Panels):
      AGuide.RF[j - AN_of_Panels] = float(FRContent[i][j])
    # Read F-vector from List of Guides file
    for j in range(2 * AN_of_Panels, 3 * AN_of_Panels):
      AGuide.F[j - 2 * AN_of_Panels] = float(FRContent[i][j])
    # Read Fitness value from List_of_Guides file
    AGuide.Fitness = float(FRContent[i][3 * AN_of_Panels])
    # Read Angles-vector from List of Guides file
    for j in range(3 * AN_of_Panels + 1, len(FRContent[i])):
      AGuide.Angles.append(float(FRContent[i][j]))

    GUIDES.append(AGuide)
  return GUIDES


# Function writing guides to a file
def Write_Guides(AGUIDES, file, AN_of_Panels):

    global Options

    sp_n = []
    sp_RF = []
    sp_F = []
```

```
    fw = open(file, 'w')
    fw.write(str(len(AGUIDES)) + "\n")

    for I in range(0, len(AGUIDES)):
        for J in range(0, AN_of_Panels):
            if AGUIDES[I].n[J] < 0:
                sp_n.append(" ")
            else:
                sp_n.append("  ")
            if AGUIDES[I].RF[J] < 0:
                sp_RF.append(" ")
            else:
                sp_RF.append("  ")
            if AGUIDES[I].F[J] < 0:
                sp_F.append(" ")
            else:
                sp_F.append("  ")
        if AGUIDES[I].Fitness < 0:
            sp_f = " "
        else:
            sp_f = "  "

    for I in range(0, len(AGUIDES)):
        for J in range(0, AN_of_Panels):
          if J == 0:
            fw.write(str(AGUIDES[I].n[J]))
          else:
            fw.write(sp_n[J] + str(AGUIDES[I].n[J]))

        for J in range(0, AN_of_Panels):
            fw.write(sp_RF[J] + str('{:.2e}'.format(AGUIDES[I].RF[J])))

        for J in range(0, AN_of_Panels):
            fw.write(sp_F[J] + str('{:.3e}'.format(AGUIDES[I].F[J])))

        fw.write(sp_f + str('{:.3e}'.format(AGUIDES[I].Fitness)) + "    ")

        for J in range(0, Options.L):

            if AGUIDES[I].Angles[J] < 0:
                sp1 = " "
            else:
                sp1 = "  "

            if AGUIDES[I].Angles[J] >= -9.9 and AGUIDES[I].Angles[J] <=
9.9:
                sp2 = "  "
            else:
                sp2 = " "

            fw.write(sp1 + str('{:3.1f}'.format(AGUIDES[I].Angles[J])) +
sp2)
        fw.write("\n")

    fw.close()
    return


# Function selecting guides to mate
def Selection(AGeneration, AN_of_Panels):
```

```
  global Options

# Create an instance of parent-guides
  AParents = TParents(AN_of_Panels)
  Alive = []
  CsGroupe = []
# Define number of individuals in the batch for looking for the parent M in
the mating operation
  Cs = 2
  # Find guides with fitness less than 1e+50 within the generation and copy
them to the Alive list
  for i in range(0, len(AGeneration)):
    if AGeneration[i].Fitness < 1e+50:
      Alive.append(AGeneration[i])

  # Choose the A-parent randomly from the Alive list
  i = 0
  while AParents.A.Fitness <= 0:
    AParents.A = AGeneration[random.randint(0, Options.N - 1)]
    i += 1
    if i > Options.N - 1:
      break

  Temp = AGeneration[0]

  # Sort the Generation of guides in ascending order of their fitness
  for i in range(0, Options.N):
    for j in range(0, Options.N):
      if AGeneration[i].Fitness < AGeneration[j].Fitness:
        Temp = AGeneration[i]
        AGeneration[i] = AGeneration[j]
        AGeneration[j] = Temp

  # Copy the Cs number of best guides to the CsGroup array
  i = 0
  j = 0
  while j <= Cs - 1:
    if AGeneration[i].Fitness > 0 and AGeneration[i].Fitness < 1e+50:
      CsGroupe.append(AGeneration[i])
      j += 1
    if i < Options.N - 1:
      i += 1
    else:
      break

  # Within the CsGroup find the guide, which is closer (has the smallest
Euclidian distance) to Parent A
  if len(CsGroupe) > 0:
    AEqDistance = EqDistance(AParents.A, CsGroupe[0])
    AParents.M = CsGroupe[0]
    if len(CsGroupe) > 1:
      for i in range(1, Cs):
        if AEqDistance > EqDistance(AParents.A, CsGroupe[i]) and
EqDistance(AParents.A, CsGroupe[i]) != 0:
          AEqDistance = EqDistance(AParents.A, CsGroupe[i])
          AParents.M = CsGroupe[i]
  return AParents

# Function for crossing over the A and M parents chosen by the Selection
operator
```

```
def Crossover(AParents, AN_of_Panels):

  global Options

  Children = TParents(AN_of_Panels)

  pc = 0.95

  pt = random.uniform(0, 1)

  if pt > 0 and pt <= pc:

    pos1 = 0
    pos2 = 0

    while abs(pos1 - pos2) < 2:
      pos1 = random.randint(2, Options.L - 3)
      pos2 = random.randint(2, Options.L - 3)

    if pos1 > pos2:
      pos = pos2
      pos2 = pos1
      pos1 = pos

    if pos1 % 2 == 0:
      pos1 -= 1

    if pos2 % 2 == 0:
      pos2 += 1

    for j in range(0, len(AParents.A.Angles)):
      if j <= pos1 or j > pos2:
        Children.A.Angles[j] = AParents.M.Angles[j]
        Children.M.Angles[j] = AParents.A.Angles[j]
      else:
        Children.A.Angles[j] = AParents.A.Angles[j]
        Children.M.Angles[j] = AParents.M.Angles[j]
  else:
    Children = AParents

  return Children


# Function for mutating the children provided by crossing over operator
def Mutation(AParents):

  global angles
  global Options

  pm = 0.1

  pt = random.uniform(0, 1)

  if pt > 0 and pt <= pm:
    allele = angles[random.randint(0, len(angles) - 1)]
    pos = random.randint(0, Options.L - 2)
    if pos%2 != 0:
      pos = pos + 1

    chooser = random.randint(1, 2)
    if chooser == 1:
```

```
      aParent = AParents.A
    else:
      aParent = AParents.M

    if allele == 0 or allele == 90:
      if aParent.Angles[pos] == 0 or aParent.Angles[pos] == 90:
        aParent.Angles[pos] = allele
      else:
        aParent.Angles[pos] = allele
        aParent.Angles[pos + 1] = -allele
    else:
      aParent.Angles[pos] = allele
      aParent.Angles[pos + 1] = -allele

    if chooser == 1:
      AParents.A = aParent
    else:
      AParents.M = aParent

  return AParents


# Function calculating the Euclidian distance between 2 guides
def EqDistance(point1, point2):

  global Options

  AResult = 0
  for i in range(0, Options.L):
    AResult = AResult + (point1.Angles[i] - point2.Angles[i]) ** 2
  AResult = AResult ** 0.5
  return AResult


# Function filling in randomly the array with stacking sequence of a guide
def FillGuideAngles(mode):

  global Options
  global angles
  AAngles = []

  if mode == "1st_Gen":
      while len(AAngles) <= Options.L - 1:
        if AAngles == []:
          AAngles.append(0)
          while AAngles[0] == 0.0:
            AAngles[0] = angles[random.randint(0, len(angles) - 1)]
#implementing Damtol design constraint

        if len(AAngles) > 0:
          if AAngles[len(AAngles) - 1] > 0.0 and AAngles[len(AAngles) - 1]
< 90.0:  #implementing +/- theta design constraintt
            AAngles.append(-AAngles[len(AAngles) - 1])
          else:
            if (AAngles[len(AAngles) - 1] == 0.0 or AAngles[len(AAngles) -
1] == 90.0) and (len(AAngles) % 2 != 0.0):  #implementing 0|90 pair
constraint
              if random.randint(0, 1) == 0.0:
                AAngles.append(0.0)
              else:
                AAngles.append(90.0)
```

```
            else:
              AAngles.append(angles[random.randint(0, len(angles) - 1)])
  else:
      for k in range(0, Options.L):
          AAngles.append(0)
  return AAngles


def createparser():
    parser = argparse.ArgumentParser()
    parser.add_argument('-EXP')
    parser.add_argument('-Gen')
    parser.add_argument('-N_of_Panels')
    parser.add_argument('-Mode')
    return parser


if __name__ == '__main__':
    parser = createparser()
    namespace = parser.parse_args()

Operate_with_Guides(namespace.EXP, int(namespace.Gen),
int(namespace.N_of_Panels), namespace.Mode)
```

B.5 "Replacer.py" - Python code for replacement of old guides with new ones

```
import argparse
import math
import random
import os
import time
from datetime import datetime
from shutil import copyfile


Stiffness = []
N_of_Panels = 0


# Class defining the Options structure of the algorithm
class TOptions:

  def __init__(self):
    self.L = 0
    self.N = 0
    self.LocalPath = ""
    self.GuidePath = ""


Options = TOptions()


# Class containing the geometrical dimensions of the structure
class TGeometry:

  def __init__(self):
    self.a = 0
    self.b = 0
    self.c = 0
```

```
Geometry = TGeometry()

# Class containing parameters of [±45₅/45]ₛ laminate of sub-structures,
which are not optimimized
class TGlobalParams:

  def __init__(self):
    self.T = 0
    self.Ex = 0
    self.Gxy = 0


GlobalParams = TGlobalParams()


# Class defining the structure of data required for a guide
class TGuide:

  def __init__(self, AN_of_Panels):

    self.Angles = FillGuideAngles()  # array of the stacking sequence
    self.n = [] # array containing number of layers for each panel
    self.RF= [] # array containing RFmin for each panel
    self.F = [] # array containing fitness for each panel
    for i in range(0, AN_of_Panels):
      self.n.append(-1)
      self.RF.append(-1)
      self.F.append(-1)
    self.Fitness = -1

# Function reading the Options of the algorithm defined by a user
def ReadOptions():

  global Options

  with open("Genetics_Guides_Opt.options", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Options.L = int(FRContent[0][1])
  Options.N = int(FRContent[1][1])
  Options.LocalPath = str(FRContent[6][1])
  Options.GuidePath = str(FRContent[7][1])
  return Options


# Function writing guides to a file
def Write_Guides(AGUIDES, file, sign, n):

  global Options
  global N_of_Panels

  fw = open(file, sign)
  if n == "Y":
    fw.write(str(len(AGUIDES)) + "\n")

  sp_n = []
  sp_RF = []
  sp_F = []

  for I in range(0, len(AGUIDES)):
```

```
    for J in range(0, N_of_Panels):
      if AGUIDES[I].n[J] < 0:
        sp_n.append(" ")
      else:
        sp_n.append("  ")

      if AGUIDES[I].RF[J] < 0:
        sp_RF.append(" ")
      else:
        sp_RF.append("  ")

      if AGUIDES[I].F[J] < 0:
        sp_F.append(" ")
      else:
        sp_F.append("  ")

    if AGUIDES[I].Fitness < 0:
      sp_f = " "
    else:
      sp_f = "  "

  for I in range(0, len(AGUIDES)):
    for J in range(0, N_of_Panels):
      if J == 0:
        fw.write(str(AGUIDES[I].n[J]))
      else:
        fw.write(sp_n[J] + str(AGUIDES[I].n[J]))

    for J in range(0, N_of_Panels):
      fw.write(sp_RF[J] + str('{:.2e}'.format(AGUIDES[I].RF[J])))

    for J in range(0, N_of_Panels):
      fw.write(sp_F[J] + str('{:.3e}'.format(AGUIDES[I].F[J])))

    fw.write(sp_f + str('{:.3e}'.format(AGUIDES[I].Fitness)) + "    ")

    for J in range(0, Options.L):

      if AGUIDES[I].Angles[J] < 0:
        sp1 = " "
      else:
        sp1 = "  "

      if AGUIDES[I].Angles[J] >= -9.9 and AGUIDES[I].Angles[J] <= 9.9:
        sp2 = "  "
      else:
        sp2 = " "

      fw.write(sp1 + str('{:3.1f}'.format(AGUIDES[I].Angles[J])) + sp2)
    fw.write("\n")

  fw.close()
  return

# The main function
def ReplaceWorstInds(AGen, AN_of_Panels, AStifFile, Ato, AE1, AE2, Amu12,
Amu21, AG12, Aro, AT, AEx, AGxy, Aa, Ab, Ac, AMode):

  global Options
  global Stiffness
  global GlobalParams
```

```
  global Geometry
  global N_of_Panels

  N_of_Panels = AN_of_Panels

  ReadOptions()

  # Reading the file with stiffnesses transferred from the upper level
  Stiffness = ReadStiffness(AStifFile)

  GlobalParams.T = AT
  GlobalParams.Ex = AEx
  GlobalParams.Gxy = AGxy

  Geometry.a = Aa
  Geometry.b = Ab
  Geometry.c = Ac

  files_n_RF_F = []

  # List_of_Guides.txt is copied from the GuidePath directory to the
current directory
  copyfile(Options.GuidePath + "List_of_Guides.txt", "List_of_Guides.txt")

  for i in range(0, N_of_Panels):
    # Generating names of files with RFs and Fitness of panels
    files_n_RF_F.append("n_RF_F_" + str(i + 1) + "_" + str(AGen) + ".txt")

    counter = 0
    while not os.path.isfile(Options.GuidePath + "n_RF_F_" + str(i + 1) +
".txt"):
      time.sleep(0.01)
      if counter == 0:
        counter = 1

    # n RF.txt is copied from the GuidePath directory to the current
directory with Generation index
    copyfile(Options.GuidePath + "n_RF_F_" + str(i + 1) + ".txt",
files_n_RF_F[i])

  if AGen == 0 and AMode == "GenerateGuides":
    # Reading guides from the List_of_Guides.txt file given by a user
    Generation = ReadGuides("List_of_Guides.txt", files_n_RF_F, Ato, AE1,
AE2, Amu12, Amu21, AG12, Aro, "F")
    # Writing guides to a new List of Guides.txt file
    Write_Guides(Generation, "List_of_Guides.txt", "w", "Y")

  if AGen > 0:
    # Reading guides from the List_of_Guides.txt file
    Generation = ReadGuides("List_of_Guides.txt", files_n_RF_F, Ato, AE1,
AE2, Amu12, Amu21, AG12, Aro, "")
    # Reading parent-guides from the Parents.txt file
    Parents = ReadGuides("Parents.txt", files_n_RF_F, Ato, AE1, AE2, Amu12,
Amu21, AG12, Aro, "F")
    # Replacing the worst guide with new A children-guide
    Generation = Replacement(Parents[0], Generation)
    # Replacing the second worst guide with new M children-guide
    Generation = Replacement(Parents[1], Generation)
    # Writing Children-guides to the Parents.txt file
    Write_Guides(Parents, "Parents.txt", "w", "Y")
```

```
    # Making a back up of the Parents.txt file with Generation index
    copyfile("Parents.txt", "Parents_" + str(AGen - 1) + ".txt")
    # Writing the guides of the new Generation to the List_of_Guides.txt
file
    Write_Guides(Generation, "List_of_Guides.txt", "w", "Y")

  # Finding the best guide in the current generation
  MinFitness = Generation[0].Fitness
  Min_i = 0
  for i in range(1, Options.N):
    if Generation[i].Fitness < MinFitness:
      MinFitness = Generation[i].Fitness
      Min_i = i
  # Writing the best guide in the current generation to the file
Best_Guides.txt
  Write_Guides([Generation[Min_i]], "Best_Guides.txt", "a", "N")


  # Making a back up of the List_of_Guides.txt file with Generation index
  copyfile("List_of_Guides.txt", "List_of_Guides_" + str(AGen) + ".txt")

  #Copying the List of Guides.txt file from the current directory to the
GuidePath directory
  copyfile("List_of_Guides.txt", Options.GuidePath + "List_of_Guides.txt")


# Function reading guides from a List of Guides.txt file
def ReadGuides(filer, files_n_RF_F, to, E1, E2, mu12, mu21, G12, ro, mode):

  global N_of_Panels
  global Options

  GUIDES = []
  FRContent = []

  counter = 0
  while not os.path.isfile(filer):
    time.sleep(0.01)
    if counter == 0:
      counter = 1

  with open(filer, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  n_RF_F_Content = []

  # Read n_RF-content from n_RF_F_files
  if mode == "F":
    for i in range(0, N_of_Panels):
      counter = 0
      while not os.path.isfile(files_n_RF_F[i]):
        time.sleep(0.01)
        if counter == 0:
          counter = 1
      with open(files_n_RF_F[i], 'r') as fr:
        n_RF_F_Content.append([line.split() for line in fr])
      fr.close()

  for i in range(1, len(FRContent)):
    AGuide = TGuide(N_of_Panels)
```

```
    AGuide.Angles = []

    fail = 0
    if mode == "F":
      for j in range(0, N_of_Panels):
        # Reading n-vector from n_RF_F_file
        AGuide.n[j] = int(float(n_RF_F_Content[j][i][0]))
        # Reading RF-vector from n_RF_F_file
        AGuide.RF[j] = float(n_RF_F_Content[j][i][1])
        if AGuide.RF[j] <= 1:
          fail = 1
        # Reading F-vector from n_RF_F_file
        AGuide.F[j] = float(n_RF_F_Content[j][i][2])
    else:
      # Reading n-vector from List_of_Guides file
      for j in range(0, N_of_Panels):
        AGuide.n[j] = int(float(FRContent[i][j]))

      k = 0
      # Reading RF-vector from List_of_Guides file
      for j in range(N_of_Panels, 2 * N_of_Panels):
        AGuide.RF[k] = float(FRContent[i][j])
        if AGuide.RF[k] <= 1:
          fail = 1
        k += 1

      k = 0
      # Reading F-vector from List_of_Guides file
      for j in range(2 * N_of_Panels, 3 * N_of_Panels):
        AGuide.F[k] = float(FRContent[i][j])
        k += 1

    # Reading Angles-vector from List_of_Guides file
    for j in range(3 * N_of_Panels + 1, len(FRContent[i])):
      AGuide.Angles.append(float(FRContent[i][j]))

    # Mode shows if the fitness values of guides in file are calculated or
not yet
    if mode == "F":
      if fail == 0:
        # If panals are not failed calculating the fitness value of a guide
        AGuide.Fitness = Fitness_func(AGuide, to, E1, E2, mu12, mu21, G12,
ro)
      else:
        # If at least one of the panals is failed the fitness value of a
guide is set to 1E+100
        AGuide.Fitness = 1E+100
    else:
      # If the fitness values of guides are calculated read them from the
file
      AGuide.Fitness = float(FRContent[i][3 * N_of_Panels])

    GUIDES.append(AGuide)
  return GUIDES


# Function reading the file with stiffnesses transferred from the upper
level
def ReadStiffness(AStifFile)

  global N_of_Panels
```

```
  FRContent = []
  AStiffness = []

  with open(AStifFile, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  for i in range(0, len(FRContent)):
    AStiffness.append([float(FRContent[i][0]), float(FRContent[i][1]),
float(FRContent[i][2])])
  return AStiffness


# Function for replacing the worst guides within the Generation by the new
children-guides
def Replacement(NewInd, AGeneration):
  Cf = 2  # - number of batches of individuals in the Replacement operation
  s = 7  # - number of individuals in batches in the Replacement operation
  # - increasing of the "s" number in the batch up to 14 leads to worth
convergance

  global Options

  ReplacePoolCf = []
  ReplacePoolS = []
  ReplaceCand = []

  if NewInd.Fitness > 0:
    for i in range(0, Cf):
      # 1 - forming the batch of candidates for replacement
      for j in range(0, s):
        ReplacePoolS.append(AGeneration[random.randint(0, Options.N - 1)])
      ReplacePoolCf.append(ReplacePoolS)

      # 2 - choosing the candidates to be replaced, which are the closest
to the NewInd

      AEqDistance = EqDistance(NewInd, ReplacePoolCf[i][0])

      ReplaceCand.append(ReplacePoolCf[i][0])

      for j in range(1, s):
        if AEqDistance > EqDistance(NewInd, ReplacePoolCf[i][j]):
          AEqDistance = EqDistance(NewInd, ReplacePoolCf[i][j])
          ReplaceCand[i] = ReplacePoolCf[i][j]

    # 3 - looking for the candidate with the worst fitness with in the
batch formed in step 2
    best = ReplaceCand[0].Fitness
    IndToDie = ReplaceCand[0]
    for i in range(1, Cf):
      if best > ReplaceCand[i].Fitness: # originally the sign was <=
        best = ReplaceCand[i].Fitness
        IndToDie = ReplaceCand[i]

    # 4 - Replacement of the individuum found in the step 3 with the NewInd
    for i in range(0, Options.N):
      if CompareInds(AGeneration[i], IndToDie) == 1:
        AGeneration[i] = NewInd
        break
```

```
  return AGeneration


# Function calculating the Euclidian distance between 2 guides
def EqDistance(point1, point2):

  global Options

  AResult = 0
  for i in range(0, Options.L):
    AResult = AResult + (point1.Angles[i] - point2.Angles[i]) ** 2
  AResult = AResult ** 0.5
  return AResult

# Function comparing the guides
def CompareInds(Ind1, Ind2):

  AResult = 1

  if Ind1.Angles != Ind2.Angles:
      AResult = 0

  return AResult

# Function filling in randomly the array with stacking sequence of a guide
def FillGuideAngles():

  global Options
  AAngles = []

  for k in range(0, Options.L):
    AAngles.append(0)

  return AAngles

# Function calculating fitness value of a guide
def Fitness_func(AGuide, to, E1, E2, mu12, mu21, G12, ro):

    global Stiffness
    global GlobalParams
    global Geometry
    global N_of_Panels

    N_of_Sections = len(Stiffness)

    lines = []
    pcomp_s = []

    F = 0
    TT = 0

    A11 = []
    A12 = []
    A22 = []
    A66 = []

    T = []
    Ex = []
    Gxy = []

    S = []
```

```
    Sx = []
    Sy = []

    x0 = []
    y0 = []

    EJx0 = []
    EJy0 = []
    GJz = []

    E_1 = E1 / (1 - mu12 * mu21)
    E_2 = E2 / (1 - mu12 * mu21)

    for j in range(N_of_Panels):
      A11.append(0)
      A12.append(0)
      A22.append(0)
      A66.append(0)
      T.append(2 * AGuide.n[j] * to)

      for i in range(0, AGuide.n[j]):
        A11[j] = A11[j] + 2 * to * (E_1 *
math.cos(math.radians(float(AGuide.Angles[i]))) ** 4 + E_2 *
math.sin(math.radians(float(AGuide.Angles[i]))) ** 4 + (0.5 * E_1 * mu21 +
G12) * math.sin(2 * math.radians(float(AGuide.Angles[i]))) ** 2)

        A22[j] = A22[j] + 2 * to * (E_1 *
math.sin(math.radians(float(AGuide.Angles[i]))) ** 4 + E_2 *
math.cos(math.radians(float(AGuide.Angles[i]))) ** 4 + (0.5 * E_1 * mu21 +
G12) * math.sin(2 * math.radians(float(AGuide.Angles[i]))) ** 2)

        A12[j] = A12[j] + 2 * to * (E 1 * mu21 *
(math.sin(math.radians(float(AGuide.Angles[i]))) ** 4 +
math.cos(math.radians(float(AGuide.Angles[i]))) ** 4) + (0.5 * (E_1 + E_2)
- G12) * math.sin(2 * math.radians(float(AGuide.Angles[i]))) ** 2)

        A66[j] = A66[j] + 2 * to * (0.5 * (E_1 + E_2 - 2 * E_1 * mu21) *
math.sin(math.radians(2 * float(AGuide.Angles[i]))) ** 2 + G12 * math.cos(2
* math.radians(float(AGuide.Angles[i]))) ** 2)

      Ex.append((A11[j] * A22[j] - A12[j] ** 2) / A22[j] / T[j])
      Gxy.append(A66[j]/T[j])

    for j in range(0, N_of_Sections):
      if j == 0:
        j1 = 0
        j2 = 1
      else:
        j1 = 2
        j2 = 3

      S.append(GlobalParams.T * GlobalParams.Ex * (2 * Geometry.a + 3 *
Geometry.c) + Geometry.a * (T[j2] * Ex[j2] + T[j1] * Ex[j1]))
      Sx.append(3/2 * GlobalParams.T * GlobalParams.Ex * Geometry.c**2 +
Geometry.c * Geometry.a * (T[j2] * Ex[j2] + T[j1] * Ex[j1]))
      Sy.append(GlobalParams.T * GlobalParams.Ex * (3 * Geometry.a *
Geometry.c + 2 * Geometry.a**2) + Geometry.a**2 * (T[j2] * Ex[j2] / 2 + 3 *
T[j1] * Ex[j1] / 2))

      x0.append(Sy[j]/S[j])
      y0.append(Sx[j]/S[j])
```

```
      EJx0.append(GlobalParams.T * GlobalParams.Ex * Geometry.c**3 +
Geometry.c**2 * Geometry.a * (T[j2] * Ex[j2] + T[j1] * Ex[j1]) - y0[j]**2 *
S[j])
      EJy0.append(GlobalParams.T * GlobalParams.Ex * (5 * Geometry.a**2 *
Geometry.c + 8 * Geometry.a**3 / 3) + Geometry.a**3 / 3 * (T[j2] * Ex[j2] +
7 * T[j1] * Ex[j1]) - x0[j]**2 * S[j])
      GJz.append(4 * Geometry.a**2 * Geometry.c**2 * ((6 * Geometry.c + 2 *
Geometry.a) / GlobalParams.Gxy / GlobalParams.T + Geometry.a / Gxy[j1] /
T[j1] + Geometry.a / Gxy[j2] / T[j2]) / (((2 * Geometry.c + Geometry.a) /
GlobalParams.Gxy / GlobalParams.T + Geometry.a / Gxy[j1] / T[j1]) * ((2 *
Geometry.c + Geometry.a) / GlobalParams.Gxy / GlobalParams.T + Geometry.a /
Gxy[j2] / T[j2]) - (Geometry.c / GlobalParams.Gxy / GlobalParams.T)**2))

      F = F + (((EJx0[j] - Stiffness[j][0]) / Stiffness[j][0])**2 +
((EJy0[j] - Stiffness[j][1]) / Stiffness[j][1])**2 + ((GJz[j] -
Stiffness[j][2]) / Stiffness[j][2])**2)
      TT = TT + T[j]

    F = (1 + math.sqrt(F)) * TT * Geometry.a * Geometry.b * ro

    return F


def createparser():
  parser = argparse.ArgumentParser()
  parser.add_argument('-Gen')
  parser.add_argument('-N_of_Panels')
  parser.add_argument('-StifFile')
  parser.add_argument('-to')
  parser.add_argument('-E1')
  parser.add_argument('-E2')
  parser.add_argument('-mu12')
  parser.add_argument('-mu21')
  parser.add_argument('-G12')
  parser.add_argument('-ro')
  parser.add_argument('-T')
  parser.add_argument('-Ex')
  parser.add_argument('-Gxy')
  parser.add_argument('-a')
  parser.add_argument('-b')
  parser.add_argument('-c')
  parser.add_argument('-Mode')
  return parser


if __name__ == '__main__':
  parser = createparser()
  namespace = parser.parse_args()

ReplaceWorstInds(int(namespace.Gen), int(namespace.N_of_Panels),
namespace.StifFile, float(namespace.to), float(namespace.E1),
float(namespace.E2), float(namespace.mu12), float(namespace.mu21),
float(namespace.G12), float(namespace.ro), float(namespace.T),
float(namespace.Ex), float(namespace.Gxy), float(namespace.a),
float(namespace.b), float(namespace.c), str(namespace.Mode))
```

B.6 Genetics_vs_gold_parallel.py" – Python code for the lower-level main optimization algorithm

```
from __future__ import print_function

import os
import shutil
import random
import time
import math
from datetime import datetime, timedelta
from shutil import copyfile

dir_path = os.path.dirname(os.path.realpath("PROBLEM"))
dir_PTS = dir_path + "_NEWPOINTS"
dir_SUM = dir_path + "_SUMMARIES"
os.makedirs(dir_PTS)
os.makedirs(dir_SUM)


# Class defining the Options structure of the algorithm
class TOptions:

  def __init__(self):
    self.L = 0
    self.N = 0
    self.MainPath = ""
    self.GuidePath = ""

# Class defining the Input structure of the algorithm
class TInput:

  def __init__(self):
    self.Generation = 0  # Generation number
    self.N_of_Panel = 0  # Panel number
    self.Stresses = [0, 0, 0, 0, 0]  # Maximum stresses extracted from FEA
results

# Initialysing the structured constant containing the Options of the
algorithm
Options = TOptions()
# Initialyzing the structured constant containing the Input for the
algorithm
Input = TInput()

# Function reading Options of the algorithm defined by a user
def ReadOptions():

  global Options

  with open("Genetics_vs_gold_parallel.options", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Options.L = int(FRContent[0][1])
  Options.N = int(FRContent[1][1])
  Options.MainPath = str(FRContent[2][1])
  Options.GuidePath = str(FRContent[3][1])

  return Options
```

```
# Function reading the Input parameters of the algorithm defined by the
Optimus from the PROBLEM file
def ReadInput():

  global Input

  with open("PROBLEM", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  Input.Generation = int(FRContent[3][5])
  Input.N_of_Panel = int(FRContent[4][5])
  Input.L = int(FRContent[5][5])
  Input.Stresses = [float(FRContent[6][3]), float(FRContent[7][3]),
float(FRContent[8][3]), float(FRContent[9][3]), float(FRContent[10][3])]

  return Input

# Reading Input
ReadInput()

# Reading Options
ReadOptions()

# Tolerance for the fitness function value??????????
Tolerance = 5

EXP = 0

# Class defining the structure of data required for a guide
class TGuide:

  def __init__(self):

    self.Angles = [] # array of the stacking sequence
    self.n = []  # array containing number of layers for each panel
    self.RF = [] # array containing RFmin for each panel
    self.F = []  # array containing fitness for each panel
    self.Fitness = -1

# Class containing information about a guide
class TChromosome:

  def __init__(self, An, AL):
    self.n = An  # Number of layers
    self.d = 0   # A special value giving a unique identifier to a
chromosome calculated by the WriteSeqFile function
    self.S = FillS(An, int(AL))  # Array containing stacking sequence
filled in by the FillS function
    self.Fitness = -1  #  Fitness value of a chromosome
    self.RF = 0  #  RFmin of a chromosome

# Class defining guides as parents to mate
class TParents:

  def __init__(self):
    self.A = TGuide()
    self.M = TGuide()

# Class defining a structured information about results of an analysis
```

```
performed by Optimus
class TResult:

  def __init__(self):
    self.RF = 0
    self.F = 0


RESULTS = []

# Function reading guides from "List_of_Guides.txt" or "Parents.txt" files
def ReadGuides(filer):

  global Options

  GUIDES = []
  FRContent = []

  counter = 0
  while not os.path.isfile(filer):
    time.sleep(0.01)
    if counter == 0:
      counter = 1

  with open(filer, 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()

  N_of_Panels = int((len(FRContent[1]) - Options.L - 1) / 3)

  for i in range(1, len(FRContent)):
    AGuide = TGuide()
    AGuide.Angles = []
    # Reading n-vector from List_of_Guides file
    for j in range(0, N_of_Panels):
      AGuide.n.append(int(float(FRContent[i][j])))

    # Reading RF-vector from List_of_Guides file
      for j in range(N_of_Panels, 2 * N_of_Panels):
        AGuide.RF.append(float(FRContent[i][j]))

    # Reading F-vector from List_of_Guides file
    for j in range(2 * N_of_Panels, 3 * N_of_Panels):
      AGuide.F.append(float(FRContent[i][j]))

    AGuide.Fitness = float(FRContent[i][3 * N_of_Panels])

    # Reading Angles-vector from List_of_Guides file
    for j in range(3 * N_of_Panels + 1, len(FRContent[i])):
      AGuide.Angles.append(float(FRContent[i][j]))

    GUIDES.append(AGuide)

  return GUIDES

# Function writing number of layers, RFmin and fitness value of a panel to
n_RF_F files, which will be transferred to GA level
def Write_n_RF(filew, GUIDES, AN_of_Panel):

  fw = open(filew, 'w')
  fw.write(str(len(GUIDES)) + "\n")
```

```
  for I in range(0, len(GUIDES)):
    fw.write(str(GUIDES[I].n[AN_of_Panel - 1]) + " " +
str(GUIDES[I].RF[AN_of_Panel - 1]) + " " + str(GUIDES[I].F[AN_of_Panel -
1]) + "\n")
  fw.close()

# Function writing Sequence files corresponding to the stacking sequence of
a guide defined by And_Vector and a particular number of layers defined by
S Vector
def WriteSeqFile(AEXP, S_Vector, Ang_Vector):

  global Options

  d = 0
  s_EXP = str(1000 + AEXP)
  fw = open("Sequence_" + s_EXP[1:] + ".txt", 'w')
  for j in range(0, Options.L):
    if S_Vector[j] != 0:
      fw.write(str(Ang_Vector[j]) + " ")
      d = d + Ang_Vector[j] ** 2 * j
  for j in range(0, Options.L): # providing the mid-plane symmetry of the
laminate
    if S_Vector[Options.L - j - 1] != 0:
      fw.write(str(Ang_Vector[Options.L - j - 1]) + " ")
  fw.write("\n")
  fw.close()
  return d


# Main optimization function
def Optimize():

  global Options
  global Input
  global EXP
  global RESULTS
  global Tolerance

  # Golden ratio
  t = 0.618

  # Minimum number of layers of a half of a laminate
  a = 2
  # Maximum number of layers of a half of a laminate defined by a user via
algorithm options
  b = Options.L

  # Reading List_of_Guides.txt or Parents.txt (depending of the generation
of the GA algorithm) file transferred from the GA level
  if Input.Generation == 0:
    copyfile(Options.MainPath + "List_of_Guides.txt", "List_of_Guides.txt")
    GUIDES = ReadGuides("List_of_Guides.txt")
  else:
    copyfile(Options.MainPath + "Parents.txt", "Parents.txt")
    GUIDES = ReadGuides(Options.MainPath + "Parents.txt")

  # Defining the number of guides read from the file with guides
  N = len(GUIDES)

  EXP = 1
```

```
  # The list containing points to be transferred to Optimus for analysis at
a particular iteration of the algorithm
  S_Generation = []

  X = []
  x = []
  Y = []
  y = []
  l = []
  RF = []
  rf = []
  Tracker = [] # Saves the history of designs ever calculated for each
GUIDE. It has 3 dimensions:
                # 1st - the number of the i-th GUIDE, 2nd - the j-th
calculated design,
                # 3rd - the array of 3 values: [X, Y, RF] = ["number of
layers", "fitness", "minimum reserve factor"]

  # Calculating the starting points of the optimization algorithm
  for I in range(0, N):
    X.append([a, b - int(round(t * (b - a) / 2) * 2), a + int(round(t * (b
- a) / 2) * 2), b])
      l.append("TBD")

  # Creating Sequence files and appending points to S_Generation list
  for I in range(0, N):
    for i in range(0, 4):
      Chromosome = TChromosome(X[I][i], Input.L)
      Chromosome.d = WriteSeqFile(EXP, Chromosome.S, GUIDES[I].Angles)**0.5
      S_Generation.append(Chromosome)
      EXP += 1

  while RESULTS == []:
    # Writing points to the NEWPOINTS file taken by Optimus to be analysed
    WriteNEWPOINTS(S_Generation)
    # Reading the results of the points' analysis
    RESULTS = ReadResults()

  # Assigning the analysis results (Fitness and RFmin) to each point
  for I in range(0, len(S_Generation)):
    S_Generation[I].Fitness = RESULTS[I].F
    S_Generation[I].RF = RESULTS[I].RF

  RESULTS = []

  # Appending the information about the points and analysis' results to the
Tracker (archive) list
  for i in range(0, len(S_Generation)):
  i = 0
  while i <= len(S_Generation) - 4:
    Y.append([S_Generation[i].Fitness, S_Generation[i + 1].Fitness,
S_Generation[i + 2].Fitness, S_Generation[i + 3].Fitness])
    RF.append([S_Generation[i].RF, S_Generation[i + 1].RF, S_Generation[i +
2].RF, S_Generation[i + 3].RF])
    i += 4

  for I in range(0, N):
    Tracker.append([])
    for J in range(0, 4):
      Tracker[I].append([X[I][3 - J], Y[I][3 - J], RF[I][3 - J]])
```

```
  locker = []  # Controls if the optimum thickness for I-th GUIDE is found.
If yes, locks this GUIDE
  counter = 0  # Counts the locked GUIDES
  left = []    # Remembers where the previous step was made to (left or
right along the X axis)

  # Assigning default values to locker and left lists
  for I in range(0, N):
    locker.append(0)
    left.append(True)

  # Starting the main cycle of the algorithm
  while counter < N:  # While at least one guide is not locked do:
    S_Generation = []

    # Calculating the search zone width
    for I in range(0, N):

      if abs(Y[I][1]) < abs(Y[I][2]):
        l[I] = X[I][2] - X[I][0]
      else:
        l[I] = X[I][3] - X[I][1]

      # Calculating the the next search points according to "Gold section"
method
      if l[I] >= 4 and locker[I] == 0:
        if Y[I][1] < Y[I][2] and RF[I][2] > 1:
          X[I][3] = X[I][2]
          X[I][2] = X[I][1]
          X[I][1] = X[I][3] - int(round(t * l[I] / 2) * 2)
          Tracker[I].append([X[I][1], 0, 0])
          Y[I][2] = Y[I][1]
          RF[I][2] = RF[I][1]
          Chromosome = TChromosome(X[I][1], Input.L)
          # indicates the search goes to the left:
          left[I] = True
        else:
          X[I][0] = X[I][1]
          X[I][1] = X[I][2]
          X[I][2] = X[I][0] + int(round(t * l[I] / 2) * 2)
          Tracker[I].append([X[I][2], 0, 0])
          Y[I][1] = Y[I][2]
          RF[I][1] = RF[I][2]
          Chromosome = TChromosome(X[I][2], Input.L)
          # indicates the search goes to the right
          left[I] = False
        # Creating Sequence files and appending points to S_Generation list
        Chromosome.d = WriteSeqFile(EXP, Chromosome.S, GUIDES[I].Angles) **
0.5
        S_Generation.append(Chromosome)
        EXP += 1
      else:
        if locker[I] != 1:
          # Locking the I-th guide and increasing the locked guides counter
          locker[I] = 1
          counter += 1

    # If at least one guide is not locked write points to the NEWPOINTS
file taken by Optimus to be analysed and read the results of the points'
analysis
```

```
    if counter < N:
      while RESULTS == []:
        WriteNEWPOINTS(S_Generation)
        RESULTS = ReadResults()

    # Assigning the analysis results (Fitness and RFmin) to each point
    for K in range(0, len(S_Generation)):
      S_Generation[K].Fitness = RESULTS[K].F
      S_Generation[K].RF = RESULTS[K].RF

    RESULTS = []

    # Appending the information about the points and analysis' results  to
the Tracker (archive) list
    J = 0
    for I in range(0, N):
      if locker[I] == 0:
        if left[I] == True:
          Y[I][1] = S_Generation[J].Fitness
          RF[I][1] = S_Generation[J].RF
          Tracker[I][-1][1] = Y[I][1]
        else:
          Y[I][2] = S_Generation[J].Fitness
          RF[I][2] = S_Generation[J].RF
          Tracker[I][-1][1] = Y[I][2]
        Tracker[I][-1][2] = S_Generation[J].RF
        J += 1

  # Deleting dublicates from the Tracker (archive) list
  for i in range(0, N):
    Tracker[i] = DelDublicates(Tracker[i])

  # Sorting the Tracker for each guide in ascending order of X (number of
layers) to find where is the best fitness value on X-axis:
  for i in range(0, N):
    if Tracker[i] != []:
      for j in range(0, len(Tracker[i])):
        for k in range(0, len(Tracker[i])):
          if Tracker[i][j][0] < Tracker[i][k][0]:
            Temp = Tracker[i][j]
            Tracker[i][j] = Tracker[i][k]
            Tracker[i][k] = Temp

  # Creating the list of the best fintess value positions for each guide
  j_min = []

  # Filling in the list by default "TBD" (To Be Defined) values
  for i in range(0, N):
    j_min.append("TBD")

  # Looking for the positions (j_min) of best fitness values on the X-axis
in the Tracker (archive) for each guide
  for i in range(0, N):
    Temp = Tracker[i][0]
    for j in range(1, len(Tracker[i])):
      if Tracker[i][j][2] > 1:
        if Tracker[i][j][1] < Temp[1]:
          Temp = Tracker[i][j]
          j_min[i] = j

  S_Generation = []
```

```
  # Defining the closest points on the X-axis to the left and to the right
from those with the best fitness
  for i in range(0, N):
    if j_min[i] != "TBD":
      if Tracker[i][j_min[i]][0] < Input.L - 2:
        dx_right = int(Tracker[i][j_min[i] + 1][0] -
Tracker[i][j_min[i]][0] - 1)
      else:
        dx_right = 0
      if Tracker[i][j_min[i]][0] > 4:
        dx_left = int(Tracker[i][j_min[i]][0] - Tracker[i][j_min[i] - 1][0]
- 1)
      else:
        dx_left = 0
      if dx_right >= 4:
        for j in range(2, dx_right, 2):
          Tracker[i].append([Tracker[i][j_min[i]][0] + j, "TBD", 0])
          Chromosome = TChromosome(Tracker[i][j_min[i]][0] + j, Input.L)
          Chromosome.d = WriteSeqFile(EXP, Chromosome.S, GUIDES[i].Angles)
** 0.5
          S_Generation.append(Chromosome)
          EXP += 1
      if dx_left >= 4:
        for j in range(2, dx_left, 2):
          Tracker.append([Tracker[i][j_min[i]][0] - j, "TBD", 0])
          Chromosome = TChromosome(Tracker[i][j_min[i]][0] - j, Input.L)
          Chromosome.d = WriteSeqFile(EXP, Chromosome.S, GUIDES[i].Angles)
** 0.5
          S_Generation.append(Chromosome)
          EXP += 1

  if S_Generation != []:
    while RESULTS == []:
      # Writing points to the NEWPOINTS file taken by Optimus to be
analysed
      WriteNEWPOINTS(S_Generation)
      # Reading the results of the points' analysis
      RESULTS = ReadResults()

    # Assigning the analysis results (Fitness and RFmin) to each point
    for I in range(0, len(S_Generation)):
      S_Generation[I].Fitness = RESULTS[I].F
      S_Generation[I].RF = RESULTS[I].RF

    # Appending the information about the points and analysis' results to
the Tracker (archive) lists
    k = 0
    for i in range(0, N):
      if Tracker[i] != []:
        for j in range(0, len(Tracker[i])):
          if Tracker[i][j][1] == "TBD":
            Tracker[i][j][1] = S_Generation[k].Fitness
            Tracker[i][j][2] = S_Generation[k].RF
            k += 1

  # Sorting the Tracker (archive) lists (for each guide) in Fitness
ascending order
  for i in range(0, N):
    if Tracker[i] != []:
```

```
      for j in range(0, len(Tracker[i])):
        for k in range(0, len(Tracker[i])):
          if Tracker[i][j][1] < Tracker[i][k][1]:
            Temp = Tracker[i][j]
            Tracker[i][j] = Tracker[i][k]
            Tracker[i][k] = Temp

      # Deleting the failed designs with RF <= 1
      while True:
        if Tracker[i] != []:
          if Tracker[i][0][2] <= 1 and len(Tracker[i]) > 1:
            del Tracker[i][0]
          else:
            break
        else:
          break

  for I in range(0, N):
    if Tracker[I] != []:


      # Assigning the values n, F and RF (corresponding to the optimal
point) to the I-th guide
      GUIDES[I].n[Input.N_of_Panel - 1] = Tracker[I][0][0]
      GUIDES[I].F[Input.N_of_Panel - 1] = Tracker[I][0][1]
      GUIDES[I].RF[Input.N_of_Panel - 1] = Tracker[I][0][2]
  return


def CleanOptimusDir():
  open("WAITFOROPTIMUS","w").close()
  os.remove("Genetics_vs_gold_parallel.running")


# Function for writing data for the next step of Optimus analysis
def WriteNEWPOINTS(ANewInds):

  global Input
  global EXP
  global dir_path

  fw = open("NEWPOINTS", 'w')

  fw.write("CONTINUE" + "\n")

  for i in range(0, len(ANewInds)):
    fw.write(str(ANewInds[i].d) + " " + str(Input.Generation) + " " +
str(Input.N_of_Panel) + " " + str(ANewInds[i].n) + "\n")
  fw.close()
  shutil.copyfile("NEWPOINTS", dir_PTS + "\NEWPOINTS" + str(EXP))

  open("WAITFOROPTIMUS", "w").close()

# Function reading results of Optimus analysis from the SUMMARY file
def ReadResults():

  global dir_path
  global RESULTS
  FRContent = []

  i = 0
```

```
  while os.path.isfile("WAITFOROPTIMUS"):
    if not os.path.isfile("TERMINATE") and os.path.isfile("LOCK.OPTIMUS"):
      i += 1
      if i == 1:
        time_start = datetime.now()
      time.sleep(0.01)
    else:
      CleanOptimusDir()
      break

  with open("SUMMARY", 'r') as fr:
    FRContent = [line.split() for line in fr]
  fr.close()
  shutil.copyfile("SUMMARY", dir_SUM + "\SUMMARY" + str(EXP))

  for i in range(0, len(FRContent)):
    AResult = TResult()
    RESULTS.append(AResult)
    RESULTS[i].F = float(FRContent[i][0])
    RESULTS[i].RF = float(FRContent[i][8])
  return RESULTS

# Function filling in a Chromosome.S array (see TChromosome class)
def FillS(An, AL):
  AS = []

  for k in range(0, AL):
    if AL - k > An:
      AS.append(0)
    else:
      AS.append(1)

  return AS


# Function deliting duplicates from the Tracker (archive) list
def DelDublicates(InpList):
  OutList = []
  for x in InpList:
    if x not in OutList:
      OutList.append(x)
  return OutList


if __name__ == '__main__':
  if not os.path.isfile("Genetics_vs_gold_parallel.running"):
    open("Genetics_vs_gold_parallel.running","w").close()
  Optimize()
  NP = open("NEWPOINTS", 'w')
  NP.write("STOP\n")
  NP.close()
  CleanOptimusDir()
```

B.7 "PCOMP_D_writer.py" - Python code for calculation of E_x and G_{xy} moduli and generation of BDF file with PCOMP card for the local FE model

```
import argparse, os, time
from datetime import datetime
from math import cos, sin, radians
```

```
# Class defining the Options structure of the lower-level algorithm
class TOptions:

  def __init__(self):
    self.L = 0
    self.N = 0
    self.LocalPath = ""
    self.GuidePath = ""

# Initialysing the structured constant containing the Options of the lower-
level algorithm
Options = TOptions()

# Function reading Options of the lower-level algorithm defined by a user
def ReadOptions():

  global Options

  for file in os.listdir(os.path.dirname(os.path.realpath(__file__))):
    if file.endswith(".options"):
      with open(file, 'r') as fr:
        FRContent = [line.split() for line in fr]
      fr.close()

  Options.L = int(FRContent[0][1])
  Options.N = int(FRContent[1][1])
  Options.LocalPath = str(FRContent[2][1])
  Options.GuidePath = str(FRContent[3][1])

  return Options


ReadOptions()

# Main function writing PCOMP cards and calculating Ex and Gxy moduli of a
laminate
def write_D_pcomp(D_file, PCOMP_file, ASEQ, AEXP, to, E_1, E_2, mu21, G12):

    global Options

    FRContent = []
    lines = []
    pcomp_s = []

    s = Options.LocalPath + "Simple_Panel_Bend_Ang.optdir\Graph1\\" +
str(ASEQ) + "\Sequence_" + str(AEXP) + '.txt'

    while not os.path.isfile(s):
      time.sleep(0.01)

    with open(s, 'r') as fr:
      FRContent = [line.split() for line in fr]
    fr.close()

    Sequence = FRContent[0]

    A11 = 0
    A12 = 0
    A22 = 0
    A66 = 0
```

```
    for i in range(0, len(Sequence)):
      A11 = A11 + to * (E_1 * cos(radians(float(Sequence[i]))) ** 4 + E_2 *
sin(radians(float(Sequence[i]))) ** 4 + (0.5 * E_1 * mu21 + G12) * sin(2 *
radians(float(Sequence[i]))) ** 2)

      A22 = A22 + to * (E_1 * sin(radians(float(Sequence[i]))) ** 4 + E_2 *
cos(radians(float(Sequence[i]))) ** 4 + (0.5 * E_1 * mu21 + G12) * sin(2 *
radians(float(Sequence[i]))) ** 2)

      A12 = A12 + to * (E_1 * mu21 * (sin(radians(float(Sequence[i]))) ** 4
+ cos(radians(float(Sequence[i]))) ** 4) + (0.25 * (E_1 + E_2) - G12) *
sin(2 * radians(float(Sequence[i]))) ** 2)

      A66 = A66 + to * (0.5 * (E_1 + E_2 - 2 * E_1 * mu21) * sin(radians(2
* float(Sequence[i]))) ** 2 + G12 * cos(2 * radians(float(Sequence[i]))) **
2)


    Ex = (A11 * A22 - A12 ** 2)/ A22/(len(Sequence) * to)
    Gxy = A66/(len(Sequence) * to)

    fw = open(D_file, 'w')
    fw.write(str(Ex) + "\n")
    fw.write(str(Gxy))
    fw.close()

    for i in range(len(Sequence)):

      if float(Sequence[i]) < -9:
        sp = '    '
      if float(Sequence[i]) > 9 or -9 <= float(Sequence[i]) < 0:
        sp = '     '
      if float(Sequence[i]) >= 0 and float(Sequence[i]) <= 9:
        sp = '      '

      lines.append('      1       ' + str('{:6.5f}'.format(to).strip('0')) +
'   ' + str('{:3.1f}'.format(float(Sequence[i]))) + sp + ' YES')

    fw = open(PCOMP_file, 'w')
    fw.write("$ Composite Material Description :" + "\n" + "PCOMP     1" +
"\n")
    fw.close()

    s_pcomp = '     '
    j = 0
    for i in range(len(lines)):
      j += 1
      if j <= 2:
        s_pcomp = s_pcomp + lines[i]
        if j == 2 or i == len(lines) - 1:
          fw = open(PCOMP_file, 'a')
          fw.write(s_pcomp + '\n')
          fw.close()
          j = 0
          s_pcomp = '     '
    return


def createparser():
    parser = argparse.ArgumentParser()
```

```
    parser.add_argument('-D_file')
    parser.add_argument('-PCOMP_file')
    parser.add_argument('-SEQ')
    parser.add_argument('-EXP')
    parser.add_argument('-to')
    parser.add_argument('-E_1')
    parser.add_argument('-E_2')
    parser.add_argument('-mu21')
    parser.add_argument('-G12')

    return parser


if __name__ == '__main__':
     parser = createparser()
     namespace = parser.parse_args()

write_D_pcomp(namespace.D_file, namespace.PCOMP_file, namespace.SEQ,
namespace.EXP, float(namespace.to), float(namespace.E_1),
float(namespace.E_2), float(namespace.mu21), float(namespace.G12))
```

Appendix C Files on electronic media