



Master's Dissertation in Mechanical Engineering on:

Digital-twins for structural assessment: model updating techniques

Patrícia Margarida Corvo Carvalho

Supervisor at FEUP: Doutor Sérgio Tavares
Co-Supervisor at Efacec: Doutor Cristiano Coutinho



Mestrado Integrado em Engenharia Mecânica

July 21, 2021

*"The only person you are destined to become
is the person you decide to be."*

Ralph Waldo Emerson

This page was intentionally left blank.

Abstract

Nowadays, finite element (FE) analysis is a broadly used tool to analyze structures and to predict their behavior under diverse loading conditions. However, when verifying the predictions against experimental data, sometimes the unsatisfying degree of correlation hinders the model's applicability. Therefore, the model that has been adopted must be adjusted, based on measured/real data, to better reflect the real structure and keep up with operation and aging conditions: this is the essence of model updating.

Inaccuracies in FE models can be caused by a variety of factors, from unsatisfactory boundary conditions to the inadequate specification of material properties or even oversimplification of the structural system under analysis. Though these factors usually account for the need to resort to model updating, one can also rely on this technique to follow the evolution of a given structure through its life cycle and identify possible aging or damage. The use of model updating techniques in the field of structural digitalization has gained strength in the past decades and several approaches have been put forward. In fact, the latest developments in this field are gaining momentum as the use of digital twins (DT) becomes more common.

Given that this is a growing area of research, a lot of different approaches can be taken to update FE models. For instance, evolutionary algorithms (EAs) are optimization tools broadly used in the context of model updating as they allow for prompt minimization of the residuals between the properties and dynamic characteristics of the FE model and the damaged structure. Inspired by nature, these algorithms combine efficiency with effectiveness, even when solving complex and highly non-linear problems.

In this dissertation, EAs are discussed, and a genetic algorithm (GA) optimization is implemented using *Ansys*, a finite element software. Additionally, a GA and a particle swarm optimization (PSO) algorithm are both implemented using Python and a dataset prepared in *Ansys*. Since model updating applied to digital twins can be modeled as either a single-objective or multi-objective optimization problem, both approaches are analyzed and their performances are compared. Considering that the ultimate goal is to apply the analyzed model updating methods to power transformers, enabling monitoring and diagnosis features of the DT, such as early damage detection, a simplification of this structure is analyzed in both cases. The initial approach to FE model updating is done in generic terms using a simple problem with synthetic data, while the second approach is based on experimental data and portrays a more complex problem.

Keywords: Power transformer; Finite element analysis; Evolutionary algorithms; Optimization; Model updating; Digital-twin

This page was intentionally left blank.

Resumo

Atualmente, o cálculo estrutural com base em elementos finitos (EF) é uma abordagem amplamente utilizada para analisar estruturas e prever o seu comportamento sob diversas condições de carregamento. No entanto, ao verificar as previsões contra dados experimentais, às vezes o grau insatisfatório de correlação limita a aplicabilidade do modelo. Portanto, o modelo adotado deve ser ajustado, com base nos dados medidos, para melhor refletir a estrutura real e acompanhar as condições de operação e envelhecimento: esta é a essência da atualização de modelos numéricos. Imprecisões em modelos de EF podem ser causadas por uma variedade de fatores, desde condições de fronteira insatisfatórias, especificação inadequada das propriedades do material ou mesmo simplificação excessiva do sistema estrutural em análise. Embora esses fatores geralmente determinem a necessidade de recorrer à atualização do modelo numérico, também se pode contar com essa técnica para acompanhar a evolução de uma determinada estrutura ao longo do seu ciclo de vida e caracterizar a sua degradação e dano. O uso de técnicas de atualização de modelos no campo da digitalização de estruturas ganhou força nas últimas décadas tendo várias abordagens sido propostas. Na verdade, os desenvolvimentos nesta área têm vindo a ganhar força à medida que o uso de gémeos digitais (GD) se torna mais comum.

Dado que esta é uma área de estudo em crescimento, várias abordagens distintas podem ser adotadas para atualizar um modelos de EF. Por exemplo, algoritmos evolutivos (AE) são ferramentas de otimização amplamente utilizadas no contexto de atualização de modelos numéricos, pois permitem a minimização dos resíduos entre as propriedades e características dinâmicas do modelo de EF e as correspondentes propriedades e características da estrutura danificada. Inspirados pela natureza, esses algoritmos combinam eficiência com eficácia, mesmo na resolução de problemas complexos e altamente não lineares.

Nesta dissertação são apresentados alguns AEs, sendo um algoritmo genético (AG) implementado usando o *Ansys*, um software de EF. Além disso, um AG e um algoritmo de otimização por enxame de partículas (OEP) são programados em linguagem Python usando uma base de dados preparada em *Ansys*. Uma vez que a atualização de modelos numéricos utilizada em gémeos digitais pode ser modelada como um problema de otimização de objetivo único ou multiobjetivo, ambas as abordagens são analisadas e os seus desempenhos são comparados. Considerando que o objetivo final é aplicar os métodos de atualização de modelos numéricos analisados a transformadores, possibilitando a monitorização e diagnóstico realizados pelo GD, como a detecção precoce de dano, analisa-se uma simplificação dessa estrutura em ambos os casos. A abordagem inicial para atualização do modelo de EF é feita em termos genéricos usando um problema simples baseado em dados sintéticos, enquanto a segunda abordagem é baseada em dados experimentais e retrata um problema mais complexo.

Palavras-chave: Transformador; Análise de elementos finitos; Algoritmos evolutivos; Otimização; Atualização de modelos numéricos; Gémeo digital

This page was intentionally left blank.

Acknowledgments

This dissertation is the apex of 5 months of study, work and dedication, encouraged and supported by a group of people and entities I would like to thank here.

First and foremost, I want to thank Engineer Sérgio Tavares, Ph.D. for his advice and availability. His assistance, through the continual and constructive monitoring of the entire work, was critical for the completion of my dissertation. I also want to thank Engineer Cristiano Coutinho, Ph.D. for sharing his knowledge and investing his time. His capacity to creatively find answers to all sorts of problems, as well as his availability and willingness to educate, were pivotal for the development of this dissertation. I would also like to thank Engineer Cassiano Linhares, who followed along the entire process, participating in all meetings and providing his timely and relevant opinion. I would also like to thank Engineer Fábio Gonçalves for sharing his knowledge while displaying great availability and initiative. Finally, I would also like to thank Efacec, in particular the Mechanical Studies R&D team, for the opportunity to work with them and for providing all the necessary resources.

I also want to thank my family, for their support, friendship and understanding, not only at this stage but throughout my entire academic career. Without your support it would be impossible to get this far. I want to thank Carolina and André for their advice and good examples. They showed me not only goals to aim for, but also ways to reach them. To Nuno, I am thankful for his affection and understanding, for his faith in me and in my abilities, for the honest conversations, and for the candid fun. I am grateful to my parents that always let me fly, reassuring me with an unfailing safety-net. They taught me to explore the world and my abilities with confidence. Finally, I want to thank my grandfather for his concern, affection and interest, always making himself present, regardless of the distance.

I am also thankful for my friends at FEUP, whom I hope to always have by my side, and who, I am blessed to say, are too many to mention in this small space. To those who always hear my laments but never doubt my potential. To those who are here in good times and bad, always making sure that the first overshadow the second.

Last but not least, I thank Jota for his love, support and patience throughout this time. Thank you for always showing me the better side of life. I also thank Olga for her consistent support, ensuring that I always felt at home.

Patricia Carvalheira

This page was intentionally left blank.

Contents

1	Introduction	1
1.1	Model updating	1
1.1.1	Structural damage detection	1
1.1.2	Digital-twin	2
1.2	Efacec	3
1.3	Proposed goals	3
1.4	Document structure	4
2	Model Updating Techniques	7
2.1	Introduction	7
2.2	Sensitivity-based methods	8
2.3	The response surface method (RSM)	8
2.4	Bayesian and Monte Carlo approaches	9
2.4.1	Monte Carlo method	9
2.4.2	Bayesian inference method	10
2.5	Computational intelligence	11
2.6	Fuzzy sets	12
2.7	Simulated annealing	14
2.8	Evolutionary computation	16
2.8.1	Evolutionary algorithms (EAs)	16
2.8.2	Genetic algorithms (GAs)	21
2.8.3	Particle swarm optimisation (PSO)	23
2.9	Machine learning techniques	28
2.9.1	Artificial neural networks (ANNs)	29
2.10	Hybrid algorithms	29
2.10.1	Hybridization between PSO and GA	29
2.10.2	Optimization of ANNs based on EAs	32
2.11	Single-objective (SOO) and multi-objective (MOO) optimization	33
2.11.1	Multi-objective evolutionary algorithms (MOEAs)	35
3	Power transformers	39
3.1	Fundamentals	39
3.1.1	Principles and equivalent circuit	39
3.2	Electric power grid	41
3.3	Power transformer operation	41
3.4	Types of transformer	43
3.4.1	On the basis of use: power and distribution transformers	43
3.4.2	On the basis of design: shell and core transformers	43
3.5	Structural elements in power transformers	45

3.5.1	Windings	45
3.5.2	Core	46
3.5.3	Tap changer	46
3.5.4	Bushings	46
3.5.5	Insulating materials	46
3.6	Typical loading scenarios for power transformers	47
3.6.1	Types of reinforcement	48
4	A case study on model updating with synthetic data	49
4.1	Problem specification	49
4.2	Ansys: model preparation and synthetic data generation	51
4.2.1	Preparing the FE model	51
4.2.2	Retrieving synthetic data	55
4.2.3	Defining parameters for model updating	55
4.3	Ansys: model updating	56
4.3.1	Defining compound output parameters	59
4.3.2	Response surface optimization (RSO)	59
4.3.3	Direct optimization (DO)	82
4.3.4	Comparison between DO and RSO	94
4.3.5	Conclusions	100
5	A case study on model updating with experimental data	101
5.1	Problem specification	101
5.2	Experimental set up and data retrieval	102
5.3	Ansys: model preparation	105
5.3.1	Defining parameters	107
5.3.2	Creating the RS	107
5.3.3	Defining the objective function	127
5.3.4	Running the optimization procedures	129
5.3.5	Implementing PSO	129
5.3.6	Implementing GA	139
5.3.7	Results and comments	147
6	Conclusions and future work	151
6.1	Main conclusions	151
6.2	Future work	154
	References	155
	Appendices	159
A	Optimization results for the RS with 125 refinement points	161
B	Optimization results for the RS with 725 refinement points	165

List of Figures

2	Model Updating Techniques	
2.1	FE model updating techniques, adapted from [2].	8
2.2	Schematic representation of the simulated annealing method, adapted from [17].	15
2.3	Generic Evolutionary Algorithm, [21].	18
2.4	Movement of a particle considering a single neighbour, [28].	25
3	Power transformers	
3.1	The power transmission process, [37].	40
3.2	Representation of a monophasic transformer composed of two windings, with a different number of turns.	42
3.3	Types of transformer on the basis of use: distribution transformer, Efacec.	43
3.4	Types of transformer on the basis of use: power transformer, Efacec.	44
3.5	Types of transformer on the basis of design: shell transformer.	44
3.6	Types of transformer on the basis of design: core transformer.	45
4	A case study on model updating with synthetic data	
4.1	Cube analysed in the case study.	50
4.2	<i>Ansys Workbench</i> : Static Structural, Parameter Set, Response Surface and Response Surface Optimization blocks.	52
4.3	FE mesh generated in <i>Ansys</i>	53
4.4	Fixed supports applied to the FE model of the cube.	53
4.5	Internal pressure applied to the 6 walls of the FE model of the cube.	54
4.6	Total deformation for the synthetic data calculated in <i>Ansys</i>	54
4.7	Von Mises equivalent stress for the synthetic data calculated in <i>Ansys</i>	55
4.8	Maximum deformation of the top wall being defined as a parameter.	56
4.9	Parameter set.	57
4.10	Design exploration options available in <i>Ansys</i>	58
4.11	Schematic representation of the conducted experiments.	58
4.12	Definition of the parameter for the SOO objective function in the parameter set.	60
4.13	RS and RSO blocks in <i>Ansys Workbench</i>	60
4.14	Initial design points created in the DOE for both RSs.	61
4.15	Relative error as a function of the number of refinement points for the RS with 525 design points.	65

4.16	Relative error as a function of the number of refinement points for the RS with 1050 design points.	66
4.17	Comparison of the maximum relative error as a function of the number of refinement points for the RSs with 525 and 1050 design points.	67
4.18	Objective definition for MOO : deformation targets.	69
4.19	Objective definition for SOO: minimization of the objective function defined under parameter P23.	69
4.20	General comparison of the sum of the squared differences between the deformations in the updated model and those in the synthetic data for RSO. .	78
4.21	General comparison of the CAE calculated for all conducted experiments using RSO.	78
4.22	General comparison of the CAEYM calculated for all conducted experiments using RSO.	79
4.23	General comparison of the sum of the squared differences between the deformations in the updated model, calculated with the FE model, and those in the synthetic data for RSO.	81
4.24	Evolution of the relative error of the average deformation value per iteration for MOO on a RS created using 1050 design points.	83
4.25	DO blocks in the <i>Ansys Workbench</i>	84
4.26	Evolution of the relative error of the average deformation value per iteration for DO using MOO with 525 design points.	90
4.27	Evolution of the relative error of the average deformation value per iteration for DO using MOO with 1050 design points.	91
4.28	Comparison of the evolution of the objective function value as a function of the number of evaluations for DO using SOO.	92
4.29	Evolution of the candidate solutions for the updated deformation of the back wall as a function of the number of evaluations for DO using MOO with 525 design points.	95
4.30	Evolution of the candidate solutions for the updated deformation of the back wall as a function of the number of evaluations for DO using MOO with 1050 design points.	96
4.31	General comparison of the sum of the squared differences between the deformations in the updated model and those in the synthetic data.	97
4.32	General comparison of the CAE calculated for all conducted experiments. .	98
4.33	General comparison of the CAEYM calculated for all conducted experiments.	98
4.34	General comparison of the sum of the squared differences between the deformations in the updated model, calculated with the FE model, and those in the synthetic data.	99
5	A case study on model updating with experimental data	
5.1	Experimental setup: pressure gauge and LVDT sensors.	103
5.2	Experimental setup: power supply and <i>LabView</i>	103
5.3	Evolution of the deformation of each wall as a function of the applied internal pressure obtained from the experimental data.	105
5.4	<i>Ansys Workbench</i> blocks for the models at various pressure levels.	106
5.5	FE mesh generated in <i>Ansys</i>	107
5.6	Fixed supports applied to the FE model of the cube.	108

5.7	Internal pressure applied to the 6 walls of one of the FE models of the cube (model subject to an internal pressure of 197.5 kPa).	108
5.8	Total deformation, calculated in <i>Ansys</i> , for the model subject to an internal pressure of 197.5 kPa.	109
5.9	Von Mises equivalent stress, calculated in <i>Ansys</i> , for the model subject to an internal pressure of 197.5 kPa.	109
5.10	Back wall thickness parameter being defined as equal between models in the <i>Parameter Set</i>	110
5.11	Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available function in RBF interpolant class <i>scipy.interpolate.Rbf</i>	123
5.12	Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available kernel in RBF interpolant class <i>scipy.interpolate.RBFInterpolator</i>	123
5.13	Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available function in RBF interpolant class <i>scipy.interpolate.Rbf</i>	124
5.14	Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available kernel in RBF interpolant class <i>scipy.interpolate.RBFInterpolator</i>	124
5.15	Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available method in the MDI class <i>scipy.interpolate.griddata</i>	125
5.16	Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available method in the MDI class <i>scipy.interpolate.griddata</i>	126
5.17	Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for the MDI class, using the 'linear' method.	127
5.18	Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for the MDI class, using the 'linear' method.	128
5.19	Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 1$).	132
5.20	Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 1.5$).	134
5.21	Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 1$).	134
5.22	Evolution of the objective function value for various combinations of c_1 and c_2 (no. particles = 50, max. no. iterations = 300, $w = 0.9$).	135
5.23	Evolution of the objective function value for various combinations of c_1 , c_2 and w , with an emphasis on the proportions between c_1 and c_2 (no. particles = 50, max. no. iterations = 300).	135
5.24	Evolution of the objective function value as a function of the population size (max. no. iterations = 200, $c_1 = 1$, $c_2 = 1.5$, $w = 0.9$)	136

5.25	Analysis of various runs of the PySwarms algorithm, using the same parameters (no. particles = 50, max. no. iterations = 300, $c_1 = 1$, $c_2 = 1.5$, $w = 0.9$)	138
5.26	Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.1). . . .	143
5.27	Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, Max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.3). . . .	143
5.28	Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.5). . . .	144
5.29	Evolution of the objective function value for various combinations of mutation probability (MP) and crossover probability (CP) (population size = 50, max. no. Iterations = 200, elite ratio = 0.01, parents portion (PP) = 0.1).	144
5.30	Evolution of the objective function value for various population sizes (max. no. iterations = 200, mutation probability (MP) = 0.3, elite ratio = 0.01 , crossover probability (CP) = 0.3, parents portion (PP) = 0.1).	145
5.31	Evolution of the deformation for the four walls as a function of the internal pressure for wall thickness of 5.4 mm for all four walls and Young modulus of 180 GPa.	149

List of Tables

4	A case study on model updating with synthetic data	
4.1	Wall thickness for the model used to generate the synthetic data.	51
4.2	Synthetic data.	56
4.3	Target deformations for each wall based on the synthetic data.	59
4.4	Upper and lower bounds for the input parameters.	61
4.5	Parameters for generating a RS with 525 design points.	63
4.6	Parameters for generating a RS with 1050 design points.	64
4.7	Relative error for each of the RSs.	64
4.8	Evolution of the maximum relative error for the RS with 525 design points and for the RS with 1050 design points.	64
4.9	Criteria used for the optimization procedure in all of the RSO experiments.	70
4.10	Details of the SOO and MOO experiments.	71
4.11	Results from the SOO experiments using the RSs with 525 design points and 1050 design points.	72
4.12	Results from the MOO experiment using the RS with 525 design points. . .	73
4.13	Results from the MOO experiment using the RS with 1050 design points. .	74
4.14	Optimal objective function value for the SOO experiments using the RSs with 525 and 1050 design points.	74
4.15	Sum of the squared differences for deformations in the MOO experiment using the RS with 525 design points.	75
4.16	Sum of the squared differences for deformations in the MOO experiment using the RS with 1050 design points.	76
4.17	Parameters for generating a RS with 125 design points and a RS with 725 design points and relative errors obtained after creating them.	77
4.18	Criteria used for the optimization procedure in the RSOs using 125 and 725 design points.	77
4.19	Candidate solution obtained from the SOO using the RS with 125 design points and corresponding deformation evaluation in the FE model.	79
4.20	Wall deformations for the candidate solution obtained from the SOO us- ing a RS with 125 design points according to each of the four RSs (using 125, 525, 725 and 1050 design points) and according to the FE model and corresponding errors.	80
4.21	Criteria for DO experiments.	85
4.22	Details of the SOO and MOO experiments using DO.	85
4.23	Results from the SOO experiments using DO with 525 and 1050 design points.	86
4.24	Results from the MOO experiment using DO with 525 design points. . . .	86
4.25	Results from the MOO experiment using DO with 1050 design points. . . .	87

4.26	Optimal objective function value for the SOO experiment using DO with 525 design points.	87
4.27	Sum of the squared differences for deformations in the MOO experiment using DO with 525 design points.	88
4.28	Sum of the squared differences for deformations in the MOO experiment using DO with 1050 design points.	89
5	A case study on model updating with experimental data	
5.1	Pressure levels and wall deformations for the experimental trial.	104
5.2	Pressure levels and wall deformations for the experimental trial.	104
5.3	Upper and lower bounds and intermediate values considered for the input parameters.	111
5.4	Young Modulus and wall thickness values considered for the validation points.	119
5.5	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available function in RBF interpolant class <i>scipy.interpolate.Rbf</i>	120
5.6	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available kernel in RBF interpolant class <i>scipy.interpolate.RBF Interpolator</i>	120
5.7	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available method in the MDI interpolant class <i>scipy.interpolate.griddata</i>	121
5.8	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available function in RBF interpolant class <i>scipy.interpolate.Rbf</i> , disregarding pressure levels 149.7 kPa and 197.5 kPa.	121
5.9	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available kernel in RBF interpolant class <i>scipy.interpolate.RBFInterpolator</i> , disregarding pressure levels 149.7 kPa and 197.5 kPa.	121
5.10	Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available method in the MDI interpolant class <i>scipy.interpolate.griddata</i> , disregarding pressure levels 149.7 kPa and 197.5 kPa.	122
5.11	Target deformations for each wall based on the experimental data.	128
5.12	Results obtained from applying PSO using a swarm of 50 particles and limiting the number of iterations to 200.	133
5.13	Results obtained from applying PSO using different swarm sizes ($c_1 = 1$, $c_2 = 1.5$, $w = 0.9$).	136
5.14	Analysis of various runs of the PySwarms algorithm, using the same parameters.	138
5.15	Results obtained from applying GA using a population of 50 individuals and limiting the number of iterations to 200.	142
5.16	Results obtained from applying GA using different population sizes (max. no. iterations = 200, mutation probability (MP) = 0.3, elite ratio = 0.01 , crossover probability (CP) = 0.3, parents portion (PP)= 0.1).	145

A Optimization results for the RS with 125 refinement points

- A.1 Results from the SOO experiments using the RS with 125 design points. . . 161
- A.2 Results regarding the wall thicknesses and Young Modulus obtained from the MOO experiment using the RS with 125 design points. 162
- A.3 Sum of the squared differences for deformations in the MOO experiment using the RS with 125 design points. 163

B Optimization results for the RS with 725 refinement points

- B.1 Results from the SOO experiments using the RS with 725 design points. . . 165
- B.2 Results regarding the wall thicknesses and Young Modulus obtained from the MOO experiment using the RS with 725 design points. 166
- B.3 Sum of the squared differences for deformations in the MOO experiment using the RS with 725 design points. 167

This page was intentionally left blank.

Listings

2.1	Generic pseudocode for the GA	22
5.1	Defining the RBF interpolation function for <code>scipy.interpolate.Rbf</code>	115
5.2	Calling the RBF interpolation function for <code>scipy.interpolate.Rbf</code>	115
5.3	Defining the RBF interpolation function for <code>scipy.interpolate.RBFInterpolator</code>	116
5.4	Calling the RBF interpolation function for <code>scipy.interpolate.RBFInterpolator</code>	117
5.5	Calling the MDI interpolant class - <code>scipy.interpolate.griddata</code>	118
5.6	PySwarms implementation	130
5.7	geneticalgorithm implementation	140

This page was intentionally left blank.

Nomenclature

Acronyms

ABC	Artificial Bee Colony Algorithms
AC	Alternating Current
ACO	Ant Colony Optimization
ANN	Artificial Neural Network
CAE	Combined Absolute Error
CAEYM	Combined Absolute Error considering the Young Modulus
DO	Direct Optimization
DOE	Design of Experiments
DT	Digital Twin
EA	Evolutionary Algorithm
EC	Evolutionary Computation
FE	Finite Element
FRF	Frequency Response Function
GA	Genetic Algorithm
LVDT	Linear Variable Differential Transformer
MOEA	Multi-Objective Evolutionary Algorithm
MOGA	Multi-Objective Genetic Algorithm
MOO	Multi-Objective Optimization
MDI	Multivariate Data Interpolation
RBF	Radial Basis Function
RSM	Response-Surface Method
SA	Simulated Annealing
PSO	Particle Swarm Optimization
SOO	Single-Objective Optimization
RBF	Radial Basis Function
RSO	Response Surface Optimization
RS	Response Surface
VEGA	Vector Evaluated Genetic Algorithm
NPGA	Niched Pareto Genetic Algorithm
NSGA	Nondominated Sorting Genetic Algorithm
NSGA-II	Nondominated Sorting Genetic Algorithm II
SPEA	Strength Pareto Evolutionary Algorithm
SPEA2	Strength Pareto Evolutionary Algorithm 2
PAES	Pareto Archived Evolutionary Strategy
PESA	Pareto Envelope-based Selection Algorithm
PESA-II	Pareto Envelope-based Selection Algorithm II

Symbols

$[M]$	Mass matrix
$[K]$	Stiffness matrix
ω	Natural frequency
$\{\phi\}$	Mode shape vector
$\{\varepsilon\}$	Error vector
P	Power [W]
P_r	Power loss [W]
V	Voltage [V]
V_r	Voltage drop [V]
I	Current [A]
R	Resistance [Ω]
L	Length [m]
A	Cross section area [m ²]
ρ	Resistivity [$\Omega \cdot \text{m}$]
E	Electromotive force [V]
ϕ_m	Magnetic Flux [Wb]
t	time [s]
N	Number of turns in a winding
t	Thickness [mm]
δ	Deformation [mm]
$\phi(r)$	Radial Basis Function
r	Radius
ε	RBF Adjustable Constant
K	Number of neighbours
σ	RBF Smoothing Parameter
c_1	Cognitive factor
c_2	Social factor
w	Inertia factor

Chapter 1

Introduction

1.1 Model updating

Nowadays finite element (FE) analysis is a broadly used tool to analyse structures and predict their behaviour under specific circumstances. However, when verifying the numerical predictions against experimental data, sometimes the unsatisfying degree of correlation invalidates the model's applicability. Therefore, the model that has been adopted must be corrected to better reflect the measured data: this is the essence of model updating, [1].

In other words, model updating can be described as a mathematical methodology that progressively updates a digital model by adjusting its parameters and assumptions so that its behavior (for example static and dynamic structural responses) gradually approaches that of the counterpart real structure that is being analyzed, [2].

Model updating techniques may be classified according to the type of measured data used which, for structural applications, can either be static or dynamic data, the latter in the form of frequency response function (FRFs) or natural frequencies and mode shapes. Besides this, model updating techniques can also be classified into non-iterative or direct methods and iterative or sensitivity methods. Sensitivity methods are more commonly used since they provide a wider choice of parameters for updating and are capable of overcoming the limitations of direct methods, as is explained in chapter 2, [3].

1.1.1 Structural damage detection

Inaccuracies in FE models can be caused by a variety of factors, from faulty boundary conditions, to inadequate specification of material and geometric properties or even oversimplification of the structural system under analysis. Though these factors usually account for the need to resort to model updating, one can also rely on this technique to follow the evolution of a given structure through its life cycle and identify possible damage. The use of model updating techniques on the field of damage detection has gained

strength in the past decades and several approaches have been presented. In fact, the latest developments in this field are gaining momentum as the use of digital twins becomes more common, [2, 3].

Structural damage can be caused by either internal or external factors and often leads to changes in the structural response, namely in structural dynamic properties, which may give way to deteriorated structural integrity and safety capacity. Therefore, keeping a close look on structural properties can be vital to guarantee early damage detection and perform timely structural maintenance operations or repairs, which in turn leads to a longer and safer structural life-cycle, [2].

In the context of damage detection, measured data must be acquired through non-destructive techniques, therefore the use of FRFs and modal properties is fairly common. These traits directly correlate with structural physical properties and can be used to characterize damage based on the premise that damage affects structural physical properties, which in turn alters structural dynamic characteristics, [2].

1.1.2 Digital-twin

The evolution of the information and manufacturing industries has led to the decline in cost of communication and sensor technology, accompanied by its miniaturization, which facilitates the integration of these technologies in virtually any product. That being said, and considering the fast advances imposed by the establishment of Industry 4.0, the interest in manufacturing products which can sense their own state as well as the state of the surrounding environment is sky-rocketing. In line with this trend, the concept of digital-twin (DT) was created shortly after the turn of the century within the aerospace industry. Nowadays, however, the concept of digital-twin is employed across numerous industries, [4].

In its essence, a DT is a comprehensive digital representation of a product which incorporates the properties, state and behavior of the physical object through models and data. The realistic models used in the DT allow for the simulation of its real counterpart's actual behavior, in the context of its surrounding environment, linking together all the relevant digital artifacts gathered. Thus, the digital twin is developed alongside its physical counterpart and remains its virtual twin across its entire life-cycle, integrating all the relevant current knowledge concerning it, [4, 5].

It is pertinent to note that some authors consider that the DT is a compilation of all the available operational data that is generated and accumulated both during product development and product use. However, considering that with the latest technology this could translate into an absurd amount of information, which would naturally be extremely hard to analyse in due time, it seems more appropriate to include only the relevant data in the models of a DT, which should be consistent with the intended purpose of that specific model, [4, 5].

With access to this sort of detailed information it is possible to ensure information continuity throughout the entire product's life-cycle, which in turn allows one to schedule timely maintenance, optimize the design, manufacturing and operation of future products based on the performance of its predecessors and derive solutions relevant to the real system. Interestingly, this seems to be the core of smart products, cultivated in a developing smart industry, [4, 5].

1.2 Efacec

Efacec is a Portuguese company founded in 1905 with the establishment of "A Moderna". In 1921, the then-named "Electro - Moderna, Lda." began activities in the electrical industry, building the required abilities to facilitate the launch of the Efacec brand in 1948, when "EFME - Empresa Fabril de Máquinas Eléctricas, SARL" was established. "ACEC-Ateliers de Constructions Électriques de Charleroi" would ultimately become the group's largest shareholder, and the firm was renamed "EFACEC - Empresa Fabril de Máquinas Eléctricas, SARL" in 1962, before selling its position in the firm. Efacec began restructuring operations and extended its international footprint in the 1960s. Today, the firm is present in over 65 countries, with economic activity on all five continents and an emphasis on exports, making it one of the largest Portuguese corporations.

Mobility, systems, and power solutions are the group's three primary areas of activity. Within mobility, Efacec provides a comprehensive variety of charging options for electric cars, including systems that make optimum use of the electrical grid infrastructure. The company's systems division focuses on energy, environment, industry, and transportation, while its power solutions division focuses on service, automation, and transformers, among other things.

Efacec has always distinguished itself by the design and production of electrical machinery, namely electrical power transformers. Transformers play an important part in civilization as a whole, allowing people to use energy. Despite the fact that their operation is highly dependent on electromagnetic phenomena, mechanical engineering is present at various phases of a transformer's project, namely for designing and optimizing components. This procedure relies on investigating operating circumstances, producing CAD models, and overall structural analysis.

1.3 Proposed goals

The main purpose of this dissertation is the exploration, development and application of model updating techniques, which may later be employed in power transformer DTs. This goal can be divided into various objectives, of which the following stand out:

- Explore various model updating techniques;

- Explore which model updating techniques offer the best characteristics to be implemented in a DT;
- Apply model updating techniques available in commercial software to a FE model based on synthetic data;
- Develop and apply model updating techniques in open source software to a FE model based on experimental data;
- Verify the quality and efficiency of the selected methods and compare their performance.

1.4 Document structure

This dissertation is structured in six different chapters, as follows:

Chapter 1 The present chapter, which is the first one in this dissertation, aims at presenting an introduction to the work that has been conducted and provides an overview of the aspects covered in the following chapters

Chapter 2 Various model updating techniques are presented, with an emphasis on optimization procedures. Computational intelligence is explored, in particular evolutionary computation (EC), with an emphasis on GAs and PSO. Other techniques, namely machine learning techniques, are also presented and possible hybridizations between approaches are discussed.

Chapter 3 Fundamental concepts regarding transformers are presented, starting with their role in a power grid, then moving on to their operation and characterization, and finally presenting their structural elements and the loading conditions they are typically subjected to.

Chapter 4 This chapter presents the first approach to model updating techniques conducted in this dissertation. Here, synthetic data is used to update a FE model created in *Ansys* and various techniques embedded in this software are documented and explored. The use of response surface optimization (RSO) versus the use of direct optimization (DO) is analysed, as well as the performance of single and multi-objective optimization approaches (SOO and MOO).

Chapter 5 In this chapter, a different experimental approach is presented. Based on experimental data, a given set of design points is analysed on a FE model created in *Ansys* and a database is generated. This database is then used to create a RS in Python on which

two optimization techniques (GA and PSO) are conducted. The process is document and results are analysed.

Chapter 6 In the final chapter, the main conclusions that can be drawn from the work developed throughout the dissertation are presented. Some future work suggestions are also mentioned.

This page was intentionally left blank.

Chapter 2

Model Updating Techniques

2.1 Introduction

As previously mentioned, FE model updating is the process of tuning a FE model to improve its ability to predict observed data from the physical structure being modeled. FE model updating techniques applied to damaged detection generally involve minimizing the residual of properties and/or dynamic characteristics, calculated between the FE model and the damaged structure. This procedure is essentially an optimization problem, where the objective function characterizes the distance between the FE model forecasts and the observed measurements and the design variables are the parameters of the FE model that need to be updated. That being said, various optimization approaches can be taken to carry out FE model updating, [2, 6].

The field of optimization is a vast one, and a panoply of approaches can be described to provide an in-depth analysis of all the options available, as can be seen in figure 2.1. These approaches can be divided into two groups: direct methods and iterative and indirect methods. The first group of approaches resorts to modal characteristics to update the FE model, providing accurate results with no need for updating parameters or iterative paradigms. However, they require accurate measurements, present a high sensibility to noise and may produce an unrealistic representation of the behaviour of the elements in the FE model mesh, resulting in mass and stiffness matrices with little physical meaning. For those reasons, they cannot be employed to damage detection, thus the need to resort to the second group: iterative and indirect methods. The methods in this group can be further divided into sensitivity-based methods, model updating using computational intelligence, response surface method (RSM) and Bayesian/Monte Carlo approaches. These methods entail physical parameter updating, leading to improved accuracy. Their specifications are explained in greater detail in this chapter, allowing for a better understanding of their advantages and limitations, [2, 6].

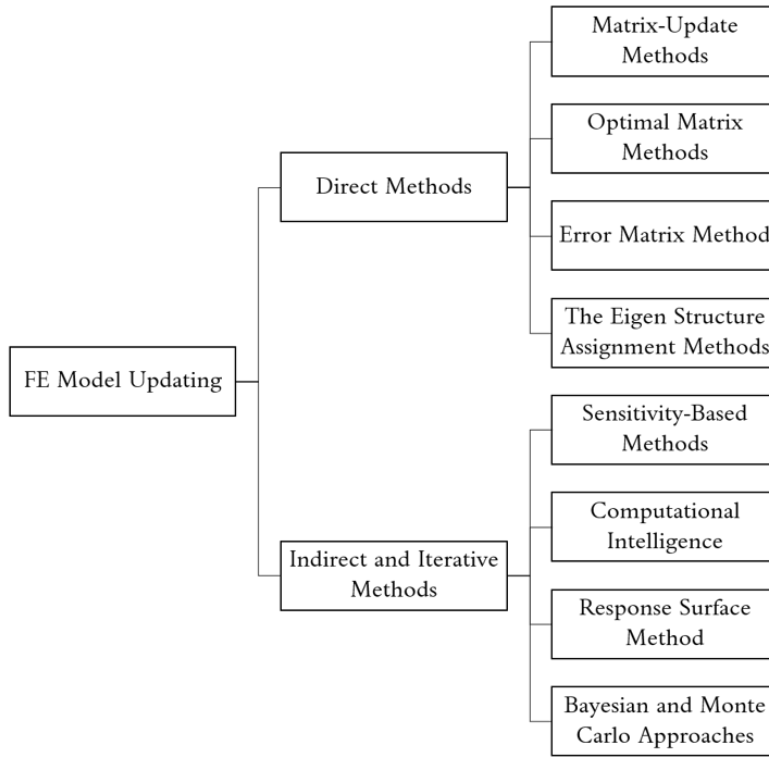


Figure 2.1 FE model updating techniques, adapted from [2].

2.2 Sensitivity-based methods

When employing sensitivity-based approaches, experimental data is assumed to be a combination of perturbations to the design variables from the intact structure's original FE model. Therefore, the optimization problem is formulated using a penalty function approach. Using this definition, the measured responses must be close to the estimated data deduced from the original FE model, limiting the applicability of sensitivity methods to small-scale shifts in the actual structure. As a result, they can only be used on structures which present minimal damage. The key idea behind sensitivity-based approaches is to quantify derivatives of modal characteristics or frequency response data, which makes the whole process computationally costly and less interesting to implement. The derivatives of either the modal properties or the frequency-response functions are calculated in these methods, [2, 6].

2.3 The response surface method (RSM)

The response surface method (RSM) is a mathematical approach for establishing a relationship between a series of fixed design variables and their responses, which is generally expressed as a multidimensional polynomial function. In the context of model updating,

the inputs are the parameters that need to be updated, and the RSM will deduce the best solution that fits the least difference between the original FE model and the evaluated responses. As a result, it is simple to implement and has a high computational efficiency. Furthermore, RSM can be used to solve dynamic model updating problems. The optimization approach is based on the response of the surface, so in comparison to a complete FE model, the optimization equation is normally simple and computationally light. Other benefits of the RSM include its ease of execution through parallel computation and the ease with which parameter sensitivity can be measured. The downside of the RSM for structural damage identification using model updating is that it uses mathematical approximations of undefined parameters, which does not accurately represent the actual damaged areas along the system. Furthermore, in large-scale systems, RSM-based FE model updating still lacks sufficient in-depth analysis, [2, 7]

Thus far, the RSM has not been widely used to solve the FE model updating problem. However, for response surface approximation, a variety of techniques have been used, namely polynomial approximation and neural networks. The response surface approximation simply creates a meta-model, the response surface, which can be used to perform optimization techniques instead of updating the FE model directly. Notice, however, that the meta-model is a simplification of the FE model and cannot be employed as freely as the FE model, as it is limited by the rationale used for the simplification of the FE model, [7].

2.4 Bayesian and Monte Carlo approaches

The Bayesian approach is a modern FE model updating technique inspired by Bayes' theorem, in which a collection of data with a probability distribution will reflect the probability distribution of a model by considering a set of data with a probability distribution. Furthermore, the parameter estimation procedure is simple to apply and provides adequate physical explanations for the effects. However, the Bayesian–Monte Carlo model updating methods face real challenges, such as the need to solve complex integrals, which results in a high computational cost. Furthermore, the original information of updating parameter periods and distributions must be understood ahead of time, [2].

2.4.1 Monte Carlo method

The Monte Carlo methods were first used in the 1940s in the context of nuclear weapon development. These methods involve experiments on random numbers and are deeply connected to experimental mathematics. To put it simply, these type of methods take a random massive sample and use it to obtain numerical solutions to a given problem. In other words, they take advantage of the randomness of the selected data and use it to solve deterministic problems. This is particularly useful when trying to solve problems

that cannot be solved analytically. All in all, the Monte Carlo methods can be used to solve any kind of problem which encompasses probabilistic interpretation, [8, 9, 6].

For instance, the Monte Carlo method can be used to solve definite integrals. However, since the solution obtained through this method is subject to the laws of chance, given its statistical nature, the accuracy of the solution will be dependent on the number of experiments undertaken. This translates in a very high computational cost, which makes the method less attractive. That being said, this method is generally used to solve problems which have a high degree of uncertainty, since problems such as the one stated above can usually be solved using more computationally economical tools, [8, 9, 6].

2.4.2 Bayesian inference method

The Bayesian inference method is a statistical method that relies on the Bayes' theorem to update the probability of a given hypothesis as more evidence or information becomes available, [10, 11].

Maximum likelihood approaches for FE model updating, which entail a process of formulating an objective function and then optimizing it, have flaws such as not providing users with confidence intervals for the solutions they produce, not presenting a philosophical rationale for the regularization principles that are used to manage the complexity of the modified model, and not being able to accommodate the FE updating problem's intrinsic ill-conditioning and nonuniqueness. The Bayesian method is then used in an attempt to fix these flaws, [11, 6].

In order to understand the definition of the Bayesian inference method, one must recall the dynamic definition of an elastic structural problem, characterized by the distributed mass, damping, and stiffness matrices. If the damping terms are ignored, the dynamic equation for the i th mode can be written in the modal domain (natural frequencies and mode shapes) as follows:

$$(-\omega_i^2[M] + [K])\{\phi\}_i = \{\varepsilon\}_i \quad (2.1)$$

where $[M]$ is the mass matrix, $[K]$ is the stiffness matrix, ω_i is the i th experimentally obtained natural frequency, $\{\phi\}_i$ is the i th experimentally obtained mode shape vector and $\{\varepsilon\}_i$ is the i th error vector. The error vector is null if the system matrices $[M]$ and $[K]$ correspond to the modal properties. If the system matrices, usually obtained from the FE model, do not match the measured modal properties, ω_i and $\{\phi\}_i$, then $\{\varepsilon\}_i$ is a non-zero vector. In maximum-likelihood methods, the Euclidean norm of $\{\varepsilon\}_i$ is minimized in order to match the system matrices to the measured modal properties, [11, 6].

Having said that, the fundamental Bayes' theorem can be stated as follows:

$$P(\{A\}|[B]) = \frac{P(\{A\} \cap [B])}{P([B])} = \frac{P(\{A\}) \cdot P([B]|\{A\})}{P([B])} \quad (2.2)$$

In the aforementioned theorem, $\{A\}$ stands for the vector of updating parameters, $P(\{A\})$ refers to the probability distribution function of updating parameters in the absence of any data, known as the prior distribution, and $[B]$ is a matrix containing natural frequencies, ω_i , and mode shapes, $\{\phi\}_i$. Note that the mass, $[M]$, and stiffness, $[K]$, matrices are functions of the updating parameters, $\{A\}$. The quantity $P(\{A\}||[B])$ is the posterior distribution function after a set of data has been seen, $P([B]|\{A\})$ is the likelihood distribution function, and $P([B])$ is the normalization factor, [11, 6].

Therefore, this method allows the combination of prior information with new information, providing a great way to analyse dynamic data. Naturally, this is a relevant technique in the field of engineering, specially when combined with FE models. The Bayesian inference model can be used while updating finite element models for damage identification, since this requires updating a model given new information. However, finding the correct probability distribution to be used in this context is often a tricky task, which is an unfortunate drawback, [10].

2.5 Computational intelligence

As stated before, FE model updating is ultimately an optimization procedure. The design variables in this case are the model's unknown parameters and the aim is to get the FE predicted data as close to the observed data as possible. Below are some of the optimization techniques typically used within the field of computational intelligence, [6, 2, 12]:

- The Nelder-Mead simplex method - a nongradient-based method for solving the unconstrained optimization problem of minimizing a certain nonlinear equation. It is a direct method which makes use of feature values in a number of places but does not try to quantify an approximate gradient at any of them.
- Sequential quadratic programming technique - solves a series of subproblems with the aim of minimizing the objective function's quadratic representation. Newton's technique is generally used when no constraints are applied.
- Fuzzy sets - applicable in contexts where information is partial or ambiguous. In a larger sense, fuzzy logic has been primarily used to analyse ambiguous or vague situations in natural language. In a narrower sense, fuzzy logic is a symbolic logic based on the concept of many-valued logic.
- Simulated annealing - a Monte Carlo approach which is used to analyze the equations of state and frozen states of n degrees-of-freedom systems with the intent of solving an optimization problem.
- Evolutionary computation (EC) - a group of techniques inspired by nature and evolution which employ stochastic algorithms to solve optimization problems.

- Machine learning techniques - a branch of techniques which employ computer algorithms that refine themselves over time as a result of their use of data and practice.
- Hybrid optimization methods - a combination of two or more algorithms.

Considering that the focus of the present work is on iterative methods, and noting that the Nelder-Mead simplex method is a direct method, it is not further discussed. On another note, the sequential quadratic programming technique presents an unattractive degree of complexity and requires full-rank constraint gradients, leading to a laborious assembly of the divergence matrix. That being said, a closer look on fuzzy sets, simulated annealing, EC, machine learning techniques and hybrid optimization techniques is provided in subsequent sections.

2.6 Fuzzy sets

As previously stated, model updating for damage assessment is an inverse problem that is susceptible to ill-posedness and ill-conditioning. As a consequence, the problem is particularly vulnerable to minor errors, which might reduce the method's dependability. Because there are various sources of uncertainty in model updating, both in terms of data and the numerical model used, it is critical to account for these uncertainties appropriately. Despite the fact that the probabilistic methods are often regarded as the most complete methodology for dealing with uncertainties, it is frequently said that it is not especially suitable for representing epistemic uncertainty. Furthermore, it is believed that adequate qualitative data for constructing a true and representative probabilistic model is rarely accessible. Parameter values, for example, are frequently unclear in the sense that they are known to be "around" a specific value or within a specific interval, and even less is known about probable interactions and dependencies between numbers. This logic has sparked the creation of several non-probabilistic uncertainty modeling methodologies, [13, 14, 15, 16].

The majority of non-probabilistic approaches rely on interval analysis, in which uncertainty about variables is represented by a specific value range, which is then propagated to interval-valued outputs of interest. Fuzzy set theory is a common extension of interval analysis. It provides a way for modeling uncertainty in circumstances when, in addition to interval limits, confidence values or degrees of uncertainty about the unknown quantities are provided. The standard binary definition of a set, where an element either does or does not belong to a set, is replaced in fuzzy set theory with a more understandable definition of sets, where membership is gradual and decided by a so-called membership function, $\mu_{\tilde{x}}$. The value of the fuzzy technique rests in the progressive description of membership, which may be interpreted in a variety of ways depending on the application. One of the

most prevalent interpretations of membership is as a measure of uncertainty or plausibility, because it provides a framework for applying fuzzy set theory to the measurement of uncertainty, [13, 14, 15, 16].

The fuzzy technique entails representing every uncertain variable x as a fuzzy number \tilde{x} , defined by a convex membership function, and then propagating these fuzzy inputs to the desired output numbers. This is commonly accomplished by discretizing the fuzzy numbers in a collection of N_{alpha} intervals, or so-called α -cuts, $[\tilde{x}]_\alpha$; as such, a fuzzy number really represents a series of nested intervals, each representing a distinct level of plausibility α . Fuzzy numbers are typically adjusted so that all membership values α fall within the range $[0, 1]$, [13, 16].

To solve a fuzzy problem, one must solve the following series of interval expressions:

$$\forall \alpha \in [0, 1] : [\tilde{y}]_\alpha = f([\tilde{x}_1]_\alpha, \dots, [\tilde{x}_N]_\alpha) \quad (2.3)$$

where an interval expression is defined as:

$$[\tilde{y}] = f([x_1], \dots, [x_N]) = \{y = f(x_1, \dots, x_N) | x_i \in [x_i], i = 1, \dots, N\} \quad (2.4)$$

The difficulty with discretized fuzzy issues (or any interval-based problem in general) is essentially in addressing the interval problem specified in equation 2.3, because the two most important solution strategies both have significant limits. The first method employs interval arithmetic, which operates directly on mathematical operators by generalizing them to interval operators. Although this method is computationally efficient, it suffers from the well-known dependency problem, in which variables that appear several times are regarded as independent of one another, resulting in massive overestimations of the derived uncertainty bounds. Other drawbacks of interval arithmetic arise in the method's actual implementation: for instance, generalizing implicitly defined operators, namely when solving a differential equation or an optimization problem, to interval operators frequently poses significant challenges, preventing the use of existing software packages.

The second method for addressing interval issues with continuous objective functions is to reformulate the interval issue in equation 2.3 as a restricted optimization problem, [13, 16]:

$$\forall \alpha \in [0, 1] : \begin{cases} y_\alpha^- = \min f(x_1, \dots, x_N) \\ y_\alpha^+ = \max f(x_1, \dots, x_N) \end{cases} \quad \text{subject to } x_i \in [\tilde{x}_i]_\alpha \quad (2.5)$$

All function evaluations become deterministic as a result of this reformulation, which means that all sorts of functions may be treated and accurate answers produced. Another advantage of the method is that, unlike the interval arithmetic technique, it is non-intrusive, which means that it may be used with any existing program for evaluating the objective function without any modifications. The related computing expense, however, is a significant downside of this method as optimization techniques often need a high

number of function evaluations. The optimization technique is most commonly used as a solution technique for equation 2.3 since the benefits of this technique generally outweigh the benefits of employing interval arithmetic. To tackle the constrained optimization problem (equation 2.5), several approaches may be used, the most prominent of which being general purpose global or local optimization algorithms and vertex approaches. Vertex methods, such as the generic and reduced transformation approaches, seek to identify the best solution to equation 2.5 by evaluating the vertices of the hypercube created by the α -cut intervals. To produce proper results, these approaches require a monotonic objective function, in which case highly efficient algorithms, for example the short transformation approach, can be produced, [13, 16].

Several enhanced vertex approaches, such as the maximum possibility search approach, have been developed in recent years to address nonmonotonic function behavior in fuzzy issues. A significant disadvantage of virtually all vertex approaches (with the exception of the short transformation approach) is that they are only relevant to situations with a very small number of unknown variables, N , because they typically need a minimum of 2^N function evaluations, [13, 14, 15, 16].

RSMS provide a means to obtain cost-efficient approximate solutions to the fuzzy interval problem when problem-specific vertex methods cannot be used, which occurs for a large number of input variables, and general optimization algorithms become too expensive, such as when a large number of expensive function evaluations are required. This is accomplished by substituting an estimated response surface for the costly function operator, which is often built by interpolation, [13, 14, 15, 16].

2.7 Simulated annealing

Simulated annealing is an optimization technique based on the natural annealing mechanism where structures such as metals recrystallize when cooled according to a set of cooling schedules. In fact, the annealing process entails heating the material until it is molten, and then gradually reducing the temperature until the system becomes ordered, ensuring thermodynamic equilibrium throughout the whole process. It is basically a Monte Carlo method since it relies on probability distributions to iteratively approach the optimal solution, [6].

Initially, the temperature is set to a very high level, and then it gradually drops throughout the process, resulting in less random variations in the solution as the temperature approaches zero. In this process a random state corresponds to a potential solution and the fundamental state, which occurs when the temperature is null, corresponds to the optimal solution. The temperature represents the characteristics under analysis, while the energy of the particles corresponds to the fitness function. Simulated annealing substitutes the current solution or state by a “nearby” random solution according to a probability determined by the difference between the corresponding objective function values and the

temperature. The probability function is defined as $e^{-\Delta E/T}$, where ΔE is the difference error between the current solution and the "nearby" solution and T is the current temperature value. This process is iterated until the optimal solution is found, [6]

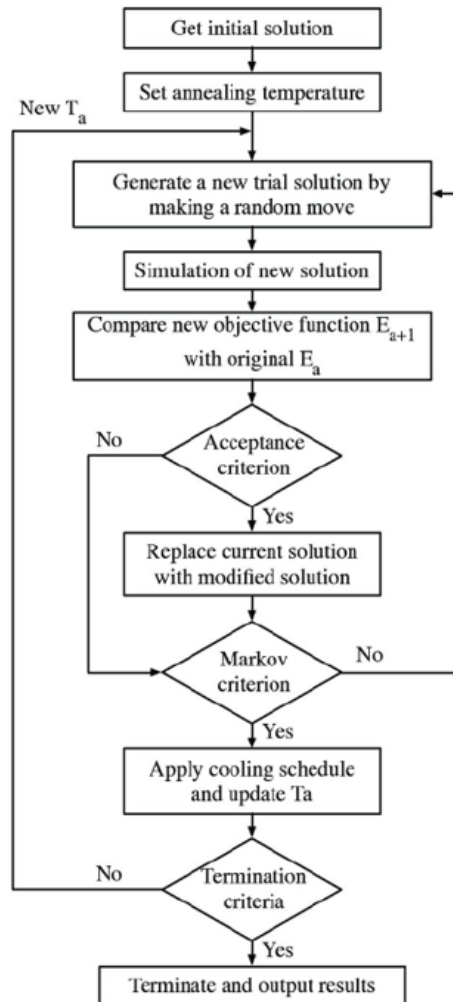


Figure 2.2 Schematic representation of the simulated annealing method, adapted from [17].

Simulated annealing, like greedy search techniques, gradually approaches the right solution, but it has the added benefit of fitness reversal. This means it may progress to a solution that is less fit than the current one, which guarantees that the solution is a global optimum rather than a local optimum. This is a significant advantage of simulated annealing over other approaches. However, it also has the disadvantage of a potentially long computational time. If the global optimum exists, simulated annealing will find it, but it may take an infinite amount of time, [6, 18].

The general benefits of using simulated annealing for optimization are, [6]:

- it can manage random structures and objective functions for any amount of implementation variables;
- it ensures that a global optimal solution is found, even though it may take an infinite number of iterations;
- it is easy to code, even for complex problems, and it is based on sound statistical grounds;
- It almost always provides a solution that is sufficiently optimal to be useful in practice.

On the other hand, the disadvantages of this method are, [6]:

- Annealing with a schedule is incredibly slow if the target function is costly to compute repeatedly. This is due to the fact that simulated annealing is basically a Monte Carlo simulation technique;
- Simulated annealing is not as effective as other methods. However, the properties of the objective function are not understood in advance for certain problems, so it is impossible to decide when to use simulated annealing for a certain class of problems;
- Simulated annealing is highly dependent on the design of the problem at hand, and it makes use of additional details regarding the system;
- It can be challenging to discern whether the optimal solution has been found;
- Deciding which temperature cooling schedule to use can be burdensome.

2.8 Evolutionary computation

Evolutionary computation (EC) encompasses stochastic algorithms inspired by nature. The search methods used in this type of techniques model natural phenomena, and some examples are particle swarm optimisation (PSO), artificial immune systems, evolutionary algorithms (EAs), ant colony optimization (ACO) or even artificial bee colony algorithms (ABC) , [19, 20].

Given their simplicity and efficiency, only EAs and PSO are discussed here. Note that PSO is particularly easy to implement and presents remarkable accuracy, while EAs depict an interestingly gradual approach to the optimal point, [19, 20, 21].

2.8.1 Evolutionary algorithms (EAs)

2.8.1.1 Historical background

Evolutionary algorithms (EAs) are population-based metaheuristics inspired by the Darwinian evolutionary theory. This theory which advocates that new individuals are

an improvement of the species based on their ancestors' competitive characteristics which allowed their survival and reproduction. Intrinsically, this methodology proposes that new and improved individuals can be developed if one can understand and model the great characteristics their parents possessed. Therefore, EAs can be used as optimization procedures and, interestingly, have great advantages over conventional gradient-based search procedures given their ability to find global optima in disjoint feasible regions for both multi-objective and single-objective functions. Furthermore, they are robust in the presence of noisy data and are able to use computing resources in a distributed and parallel manner, [22, 23].

According to [21], there are three branches within EAs which emerged independently:

1. evolution strategies
2. evolutionary programming
3. genetic algorithms (GAs)

However, according to some authors, genetic programming is also considered an independent branch. Anyhow, nowadays, GAs are the most popular type of EA and therefore these are the branch explored here in greater detail, [22, 23, 24].

2.8.1.2 General algorithm and definition

In the context of EAs, the individuals subjected to evolution are in fact the solutions to the problem at hands. These may be more or less efficient, thus the need to undergo evolution. If there are several individuals being handled simultaneously, one must refer to them as a population. Each iteration of the algorithm will create new generations. The algorithm comes to a stop when a termination criterion is met, [21].

As can be seen in figure 2.3, the EA starts with a population of individuals that represent solutions to the problem (search points). This first population is usually created at random. Each individual is evaluated with a fitness function in order to access the quality of the solution it provides. The fitness function is usually the objective function. Afterwards, the fittest individuals are selected and some operators such as crossover, mutation, selection, and reproduction are applied. The new individuals' fitness is evaluated and only the fittest among all individuals are selected, thus generating a new population. Although some individuals are eliminated, the population size does not change since new individuals are created with each iteration. This iterative process is repeated until the termination criterion is met, [25].

It is interesting to notice that, with each iteration, the individuals (or search points) which present worse fitness values are eliminated, so all new individuals derive from the fittest predecessors. This is the essence of the evolutionary algorithm and the main difference between algorithms is usually the type of operators used, [19].

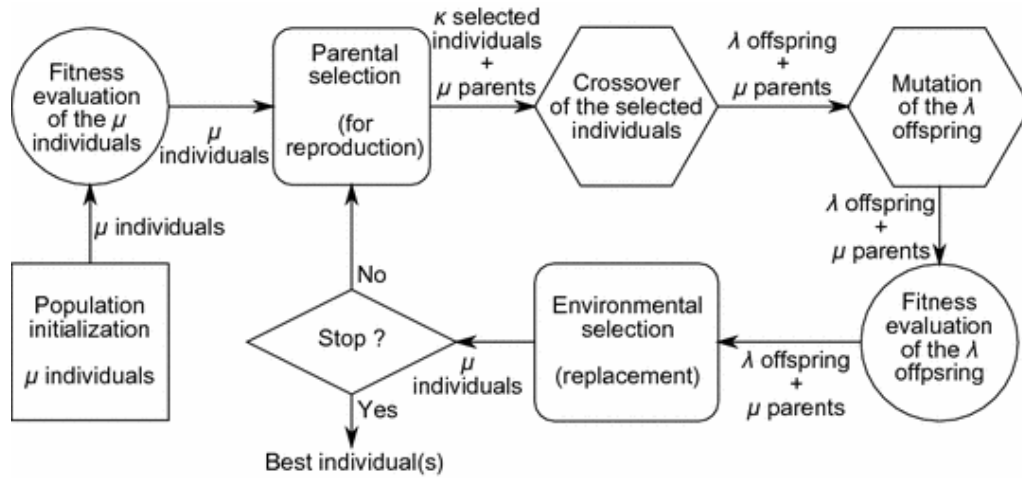


Figure 2.3 Generic Evolutionary Algorithm, [21].

Although the convergence provided by the survival of the fittest approach is fundamental for the effectiveness of the algorithm, diversity must always be preserved to ensure that a considerable portion of the domain is evaluated and local optimal points aren't mistaken for global optimal points. The latter situation tends to happen when no actions are taken to avoid the population to be filled with copies of the best individual. In other words, the algorithm shouldn't work under the sole action of selection operators, but instead incorporate variation operators as well, [21].

2.8.1.3 Operators

As mentioned before, two types of operators can be applied in EAs: selection operators and variation operators. Each of these operators affects the optimization in a distinct way, so regulating their presence in the algorithm can substantially shift its course, [25].

Selection operators are divided into selection, which determines the amount of times any given individual will reproduce in a generation, and replacement, which determines which individuals will have to disappear from the population in each generation to ensure that there are no changes to the size of the population. It is relevant to emphasize that fitter individuals are more often selected to reproduce, [21].

On the other hand, variation (or search) operators provide the algorithm with the capacity to scout for better solutions than the ones represented in the population. In other words, these operators can combine individuals or disperse the population to ensure that the optimization search covers a considerable portion of the domain. These too are divided into two categories: mutation, which simply modifies an individual to create a new one, and crossover, which generates one or more offspring from combinations of two parents. Notice that, in the biological sense, crossover can only involve two parents, however some proposals for the involvement of more than two parents in EAs have been put forward, allowing for a peculiar and more concise exploration of the search space, [21].

The aforementioned operators provide diverse techniques to promote the evolution of the population, however the way in which an individual is modified is highly dependent on the structure of the solution it represents. So, it is not possible to design a universal approach to these operators, as they always depends on the problem under analysis. The operators used, in particular variation operators, are inescapably related to the representation of the solutions in the search space. Consequently, when choosing a particular representation, it is fundamental to specify and define the variation operators to be used, as they intrinsically depend on it, [21].

Selection operators: Selection pressure and Genetic drift

As mentioned above, EAs select individuals for reproduction based on their fitness. So, the fittest individuals will reproduce more often than others and eventually replace them completely. In fact, if there is no contribution from variation operators, eventually the best individual will dominate the reproduction chain and soon the whole population will consist of copies of this super-individual. This phenomenon can be quantified by the takeover time, which is the number of generations necessary to fill a population with copies of the best individual under the sole action of selection operators. Selection pressure inversely correlates with takeover time, so it is greater when takeover time is shorter, [21].

When an EA displays a high selection pressure, then it is at risk of premature convergence. This means that a super-individual, which may be a local optimal point, takes over the population and deviates the solution from the global optimal point. Since this super-individual creates what could be described as a gravitational pull, the search space becomes limited to the area surrounding it, leaving out potentially better solutions which may be outside this limited search space, [21].

Another interesting phenomenon that happens in EAs (as well as in nature) is genetic drift. Given the stochastic nature of EAs, and in the absence of variation operators, a population formed from different individuals with similar fitness, may converge towards a state where all the individuals are identical. This phenomenon can also be quantified by the time required to obtain a homogeneous population. Interestingly, it has been noted that the time of convergence towards this plateau becomes longer as the population size increases, [21].

Genetic drift also correlates with selection pressure, the first prevailing when the latter is lower. Both these situations are characterized by a lack of diversity, however genetic drift may converge to a fitness value far from the optimum, since it is independent of the individuals' fitness, [21].

All in all, to avoid malfunctions in EAs one must guarantee an adequate population size and a selection pressure that is neither too weak nor too strong, preferably combined with some sort of variation operator, [21].

Types of Selection

Within selection one can choose from a variety of approaches, namely, [21]:

- *Proportional selection* - The expected number of times an individual is selected to generate offspring is proportional to its fitness
- *Tournament selection* - A random number of individuals are drawn from the population and only the fittest of those are selected for reproduction; these tournaments can be conducted several times with different sets of individuals
- *Truncation selection* - The n best individuals of a population are chosen for reproduction; n is a parameter chosen by the user
- *Environmental selection* - Determines which individuals in a given generation, from among both offspring and parents, will constitute the population in the next generation

It is relevant to note that proportional selection can be applied through various methods, of which the most common are Roulette Wheel Selection (RWS) and Stochastic Universal Sampling (SUS). Unfortunately, RWS displays high variance, leading to the possibility of good individuals not being selected, which in turn promotes genetic drift. This effect can be minimized if the population size is sufficiently large, but nevertheless it should not be taken lightly. On the other hand, SUS displays weaker variance than RWS, leading to less frequent genetic drift. Furthermore, proportional selection exhibits an almost non-existent selection pressure when approaching the optimum in the case of continuous functions. This behaviour can be reduced if fitness function scaling techniques are applied, though it becomes evident that managing selection pressure when applying proportional selection is a great challenge, [21]

As for environmental selection, it can also be conducted via several distinct methods, namely, [21]:

- *Generational Replacement* - Parents are completely replaced by their offspring with each generation
- *Replacement in the Evolution Strategies* - A truncation selection of the best offspring forms the population for the next generation
- *Steady-State Replacement* - In each generation a small number of offspring (one or two) are generated and they replace a smaller or equal number of parents to form the population for the next generation
- *Elitism* - The individual with the best fitness (or a group of individuals with the best fitness) are preserved in the population through the generations, thus ensuring

that the fitness of the best individual does not decrease from one generation to the next

All in all, the choice of the right approach and method should always take into consideration the problem under investigation since a great fit for one problem might not be the best fit for a different problem, which is to say it might lead to overfitting, [21].

2.8.1.4 Fitness function

As mentioned before, a fitness function allows one to quantify the quality of a candidate solution to a given problem. This function can sometimes coincide with the objective function, although that is not always the case. Noticeably, not only should the operators fit the problem, but the fitness function should also suit the chosen operators. Naturally, when creating a suitable fitness function, the representation chosen should be considered, as well as the nature of the variation operators, otherwise the information it provides will not be satisfactory in the process of approaching the optimum. Besides these parameters, the fitness function must also satisfy specific criteria which may be related to its complexity, to the constraints of the problem or even to the adjustment of the selection pressure during the evolution. The complexity of the fitness function in particular should be considered with extreme scrutiny since it will affect the agility of the algorithm and the amount of computing power it requires, given that the fitness function is calculated for all the potential solutions, [21].

Generally, when analysing real-world problems, the evaluation of the fitness function consumes the greatest amount of computing power during an evolutionary optimization. In the case of damage detection, which involves the FE method, this situation is particularly acute, given that this method is computationally intensive. Therefore, it is interesting to explore strategies which help reduce the computational time and cost. These strategies can range from parallel computing to approximate calculation of the fitness function with progressive refinement of the approximation (also defined as reduced order models (ROMs)). In the latter scenario, the difficulty resides in determining when the fitness function should be refined in order to avoid premature convergence to false solutions generated by the approximations. Besides these strategies, one can also resort to the use of tournament selection or a ranking selection which do not require the precise values of the objective function, since only the ranking of the individuals is significant, [21].

It is also relevant to note that the evaluation of the fitness function is generally part of an evaluation routine, which previously decodes the structure chosen to represent a given solution, [26].

2.8.2 Genetic algorithms (GAs)

As our knowledge of the world evolves, so do our theories that explain natural phenomena. In this context, Darwin's theory of evolution has been enriched by the our growing

knowledge on genetics, giving way to a new theory: Neo-Darwinism. Inspired by this new theory, genetic algorithms (GAs) were first presented to the world in 1975 by Holland, [21].

The classical GA operates under the same generic specifications of the common EA, as can be seen in the pseudocode presented below. However, unlike EAs, GAs refer to candidate solutions as chromosomes instead of individuals and these are usually vectors composed of decision variables. In fact, GAs apply the concept of genotype-phenotype found in natural genetics. In this context, the decision variables are called genes, as these define the characteristics that will be manifested by the problem under investigation - in natural genetics, these characteristics are what is called the phenotype. In other words, the phenotype expresses the natural representation of a given solution obtained after decoding the genotype. The genotype is the complete set of genetic material an organism possesses, which in the context of GAs would be a candidate solution to the problem, that is, a combination or set of genes, and they are the elements that are manipulated by the GA. Historically, GAs worked with solutions represented by binary strings and that is still typically the case nowadays, [21, 19, 26].

Listing 2.1 Generic pseudocode for the GA

```
begin
  t = 0;
  initialize P(t);
  evaluate individuals in P(t);
  while termination condition not satisfied do
    begin
      t = t+1;
      select_repro C(t) from P(t-1);
      recombine and mutate structures in C(t)
      forming C*(t);
      evaluate structures in C*(t);
      select_replace P(t) from C*(t) and P(t-1);
    end
  end
```

Noticeably, in its original form, the GA generated only one or two offspring per generation, maintaining the number of individuals in the population. Aside from this, when they were first proposed, GAs had three distinctive features which differentiated them from other EAs, [26]:

1. the representation used consists of bitstrings
2. the method of selection is proportional selection

3. the primary method for producing variation is crossover

As EAs evolve and alternative methods and representations are introduced, these distinctive features become faint. However, most GAs are still inspired by this original essence, [26].

So, given the information provided, one can safely say that a common GA requires two things, [26]:

1. a genetic representation of the domain
2. a fitness function to evaluate each candidate solution

Additionally, it is relevant to note that GAs can implement variation through a range of methods when practicing crossover. One of these methods is to have one global population where any given individual can be subjected to crossover. Another method, usually referred to as the island model (alternatively, the migration or coarse-grain model), holds separate subpopulations where selection and crossover take place in isolation from the other subpopulations. Nonetheless, now and again an individual from one of the subpopulations may migrate to another subpopulation, allowing for information to be shared among subpopulations. A third method, usually referred to as the neighborhood model (alternatively, the diffusion or fine-grain model), maintains overlapping neighborhoods. The neighborhood where selection (for reproduction and replacement) happens is restricted to a particular region. In this context, the definition of neighborhood and the interactions that occur between neighborhoods will depend on the neighborhood topology used, [26].

As a final note on GAs, it is interesting to point out that Holland's initial work in this area was directed towards the broader context of exploring GAs as adaptive systems instead of optimization algorithms. In fact, one can argue that, although evolution can be seen as a form of optimization, it does not particularly aim at a certain target or destination and could thus be perceived as an aimless optimization since it is an opportunistic process which operates in an ever changing environment. In that sense, GAs as a simulation of evolution would not be solution solving tools per se. Nonetheless, this perspective on GAs does not, by any means, intend to devalue the usefulness of GAs as tools for function optimization, but instead aims to point out the differences that exist between optimization and adaptation and how these different goals affect the requirements of the corresponding algorithm, [26, 27].

2.8.3 Particle swarm optimisation (PSO)

Particle swarm optimisation (PSO) was presented to the world in 1995 by James Kennedy and Russel Eberhar. Equally inspired by nature, this technique mimics the social networks of birds and fishes to carry out the exploration of a given search space in an attempt to identify an optimal point. In other words, this optimization method is based on cooperation without selection, [28, 29].

To better understand the idea behind PSO, consider how a flock of birds interacts when searching for food. They start by moving randomly within a given area. Then, as each member of the flock finds a promising spot, it informs others that might be close by. Gradually, the whole flock learns these bits of information and the best spot starts attracting more and more birds. The interesting aspect to exploit in this analogy is the social interactions between birds rather than solely individual abilities, [28].

2.8.3.1 Particles: movement and communication

In the context of PSO, each singular entity is designated as a particle and a set of particles which interact with each other is called a swarm. At first, the particles are placed within the search space and they evaluate the objective function at their current location. Then, based on the knowledge of their own current and best locations and on the information provided by other close-by particles, they move across the search space. Each new iteration begins after all particles have moved. Sooner or later the swarm will converge to an optimal point, just like the aforementioned birds concurred to the same spot, [29, 28].

Interestingly, some variants of the algorithm consider three different types of particles: explorers, memorizers and random number generators (RNG). In most algorithms, the first two particles mentioned are considered a single composite particle which incorporates the capacity to search for different points, provided by the explorer, and the capacity to memorize the best points found so far, provided by the memorizer. On the other hand, most algorithms incorporate a single RNG, which generates numbers according to a uniform distribution in a given interval, but they may also incorporate a second RNG, which generates numbers according to non-uniform distributions. These numbers are used to define the velocity of each particle, as is explained in the following paragraphs. Additionally, the mentioned particles communicate and exchange information between them. Noticeably, if an information link exists between two composite particles, the beginning of the link is taken to be the memory part of the first particle, and the end of the link the explorer part of the second particle, [28].

In order to effectively perform the PSO, each composite particle is made up of three N -dimensional vectors, where N is the dimension of the domain. These three vectors are the current position, \vec{x}_i , the previous best position, \vec{p}_i , and the velocity, \vec{v}_i , [29].

Generally, the current position, \vec{x}_i , is a set of coordinates which characterizes a point in space. However, it may also be defined as a vector of bitstrings. At the end of each iteration of the algorithm, the current position of each particle is assessed as a potential solution to the problem, which is to say that the objective function is evaluated at that point. If the position of a certain particle proves to be better than any of the positions evaluated before, then its coordinates are stored in the previous best position vector, \vec{p}_i . Additionally, the best value of the objective function found thus far is stored in a variable usually called p_{best_i} (which stands for “previous best”). This value is used for comparison

on ensuing iterations. The goal is to go on finding better positions and updating \vec{p}_i and p_{best_i} . The next point the particle must travel to is determined by adding the velocity coordinates, \vec{v}_i , to the current position, \vec{x}_i . The velocity, \vec{v}_i , is in turn determined according to the previous best position vector, \vec{p}_i , and the previous best position vector of neighbour particles. By adjusting the velocity one can effectively redirect the course of a given particle and therefore ensure that a certain area is examined by that particle, [29].

The process described above is represented in figure 2.4, where the movement of a given particle is affected by a single neighbour and the velocity vector is calculated using a linear combination. The three tendencies (maintaining the current velocity, approaching the previous best position and approaching the previous best position of a neighbour particle) are represented by three vectors, which are added, [28].

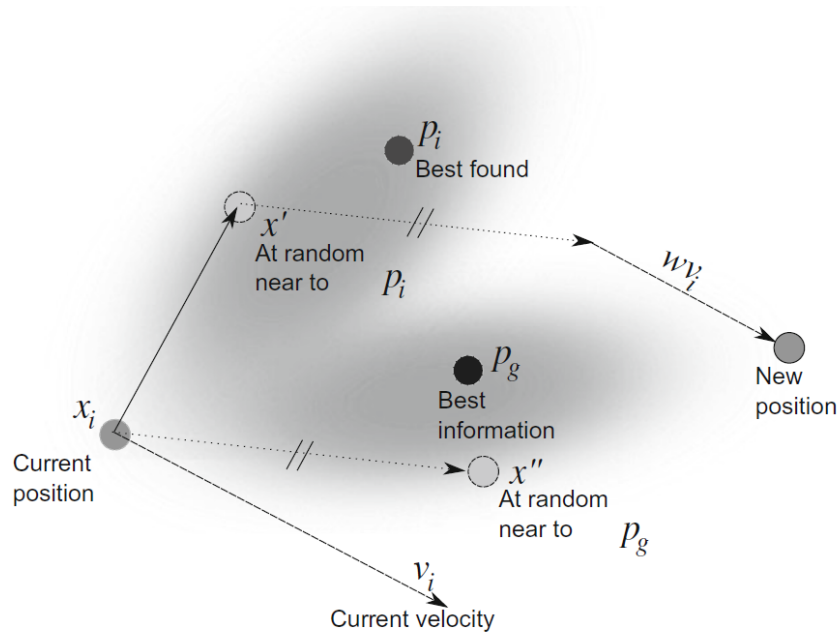


Figure 2.4 Movement of a particle considering a single neighbour, [28].

Although the basic idea behind the algorithm is fairly simple to grasp, the core of it, that is, the communication between particles, can be a bit more intricate. The communication structure or topology can be seen as a social network, and it frequently consists of bidirectional connections between pairs of particles, [29]. In this context it is useful to define the neighborhood of a particle (in a given time step), which is made up of all the particles which share an information link with it. The ensemble of all neighborhoods is the topology of the swarm. For instance, if there can be a link between any of the particles in the algorithm, then the topology is considered global. Any other topologies are said to be local, [28].

Notice that when each particle is influenced by its neighbors, several parallel searches can be executed at the same time, allowing for a broader exploration of the search space

and avoiding the misguided convergence to a local optimum instead of the global optimum. Though this strategy takes more time to complete and demands more computational effort, it is by far more accurate than having all particles influenced by the best particle in the whole swarm. The latter option converges quickly but is far more vulnerable to converging to local optima. An interesting approach to benefit from both these strategies would be to implement a dynamic topology which starts by searching within small neighborhoods and slowly increasing the sizes of these until they overlap and merge, and eventually connect all the particles in the search space in a single global neighborhood. Alternatively, the neighborhoods themselves may also be dynamic and defined by proximity in the search space and through the course of the run the number of neighbors may increase, leading to a situation where all particles become neighbors, [29].

In any given iteration, the existing relationships can be represented by a valued graph which relates particles with links between particles. Each link is defined by three components: its source, its sink and a probability value. More often than not, the topology is represented by a square matrix, T , of size n , where n is the swarm size and $T(i, j)$ the probability of particle i informing particle j . Most variants consider the probability to be either 0 or 1, and it is always assumed that a particle informs itself. That being said, for any given PSO problem the total number of possible topologies is 2^{n^2-n} , [28].

All in all, the algorithm performs quite differently depending on the topology implemented; these differences depend on the function tested, and unfortunately there is no evidence to conclusively suggest that any particular topology is generally better than any other. However, it is safe to state that the significance of the social network and the effects of its topology depend on the fashion of the interactions of the particles, [29].

2.8.3.2 Mechanisms, Strategies and Population dynamics

Management of particles

Contrary to GAs, PSO in its original form does not require mechanisms to create or destroy particles. Therefore, the only mechanism required to manage particles is the mechanism which creates them at the start of the algorithm. Notice that the size of the population is usually set empirically and is influenced by the dimension and perceived difficulty of the problem. The most classical method for this creation is to simply apply a random position inside the search space, as well as a random velocity. Even though more sophisticated methods could be used, their influence in the performance of the method usually fades after a few iterations. It is also relevant to note that, if the algorithm reaches some sort of stagnant phase, the mechanism used to create the particles could be used to create a new set of particles to replace the ones used up until that moment, [28, 29].

Management of information links

In order to function properly, a PSO must guarantee three essential functions for the information links: creation, deletion and valuation (assigning a probability). In many cases the swarm size is constant and the topology remains fixed through the whole process of optimization, therefore information links may be created just once at the beginning of the process. On the other hand, if this method does not allow the efficient optimisation of a particular problem, then the PSO can resort to some form of adaptive topology, which may be based on mathematical criteria or be inspired by social and biological behaviors, [28].

Management of particle movement

The algorithm of the PSO determines that each particle's movement is influenced by three factors, [28]:

- i. its own velocity;
- ii. the tendency to move towards the best positions memorized by its neighbors;
- iii. the tendency to move towards the best know position.

In order to compute the exact movement of a given particle, five steps must be followed, [28]:

1. Select the particles within the neighborhood of the particle which will be taken into account. The best one alone may be used, but any number of particles may be considered
2. For each neighbor particle considered, determine a point near its best memorized position and define a virtual movement towards that point
3. Combine all virtual movements considered. The current velocity should also be taken into account, though it should be of lesser influence
4. Apply the determined movement to the particle
5. In the event of the particle flying outside the search space, a confinement mechanism may be applied

Confinement and constraints

Since the search for an optimum is bonded by a search space, PSO is always subject to constraints. These constraints may be applied directly to a certain variable or be given as a relation between variables. In any case, in the event of a particle disrespecting a given constraint (that is, in the event of the particle flying outside the search space), one of two options can be taken, [28]:

- Let the particle fly and disregard the new position, allowing it to return to the search space at its own pace. This return is a result of the attraction of neighbor particles' best memorized position and its own previous best position
- Apply an immediate or progressive confinement method

The nature of confinement methods may be given by any iterative algorithm, but the most straightforward method would be to stop the particle at the frontier of the search space and either set its velocity to zero or invert its direction, which may be done at random, [28].

Noisy functions

When analysing real-world problems, it is common to come across noisy functions. Unlike the case of dynamic problems where the fitness function evolves over time, in the case of noisy problems the fitness function remains the same through the iterations. However, the fitness values associated to each evaluation may fluctuate, which could mislead the optimization search. Interestingly, the PSO algorithm responds quite well to this sort of input, remaining effective and sometimes even benefiting from it to avoid getting trapped in local optima. In any case, some noise-resistant variants of the algorithm have emerged in literature and these perform exceedingly well, making noisy data an irrelevant issue, [29].

2.9 Machine learning techniques

Machine learning is the study of the design, creation, and implementation of algorithms that allow computers to learn from data. Machine learning's key goal is to learn how to recognize complex patterns and make intelligent decisions based on data independently. As a result, machine learning is deeply entwined with fields like probability theory, statistics, and computational intelligence, [6].

Machine-learning techniques are divided into the following groups, depending on the technique's goal, [6]:

- *Supervised learning*: defines a mathematical function that maps inputs to outputs. Multi-layer perceptron and support vector machines are two examples of these techniques.
- *Unsupervised learning*: models a series of inputs without knowing what the outputs will be.
- *Reinforcement learning*: is a method for learning how to respond to a specific observation in the environment by ensuring that each behavior has an effect on the

environment and that the environment provides reinforcement in the form of incentives and punishments.

2.9.1 Artificial neural networks (ANNs)

Similarly to EAs, artificial neural networks (ANNs) are a methodology inspired by nature. A neural network is a data-processing model based on how biological nervous systems, such as the human brain, process information. That being said, a neural network is a type of computer program that models the relationship between input and output parameters. In its essence, ANNs are a method of supervised learning, [6, 7].

An ANN is made up of a group of processing units, also known as neurons, which are linked together. Each neuron is a transfer mechanism, and it can be represented as an oriented graph. Note that a neuron is a nonlinear element with many inputs and a single output. The architecture of a neural network is therefore dictated by the network's interactions and the neurons' transition functions, [7].

In ANNs, the learning process is normally applied by examples; this process is also known as 'training,' since the learning procedure is accomplished by iteratively changing the connection weights. There are three kinds of ANN learning processes: supervised, unsupervised, and reinforcement learning. The precise comparison of the current output and the predicted output is the foundation of supervised learning. Reinforcement learning is a form of supervised learning in which the only criterion is whether the output is valid. Unsupervised learning relies solely on the input data's similarity. The learning rule, which determines the weight update rule, is the heart of a learning algorithm. Delta law, Hebbian rule, and competitive learning rule are three common learning rules, [7].

As stated before, machine learning algorithms are basically methods for determining parameter values that appear to be likely based on the results. The data is normally divided into preparation, validation, and research sets during the learning process. In ANN, this is used to pick the model to make sure the trained network isn't skewed against the data it's seen during testing, [6, 7].

2.10 Hybrid algorithms

2.10.1 Hybridization between PSO and GA

Although both GAs and PSO algorithms perform in a satisfactory manner, it is possible to combine their characteristics to create a hybrid algorithm with better attributes. Notice that, given the fact that the algorithm's performance is highly dependant on the problem under analysis, one may find it hard to determine which algorithm, hybrid or purebred, is the absolute best choice in all scenarios. Therefore, the conclusions drawn from a particular comparative analysis should not be extrapolated to the general case without careful consideration.

In order to explore the vast possibilities of combining GAs with PSO algorithms, two articles were analysed, [30] and [31]. The first article analyses the performance of PSO and evolutionary optimization in the search for the optimum in well known functions with varying dimension and initial population characteristics. After drawing some conclusions regarding the performance of each method, a few hybrid algorithms are proposed, though they are not tested. On the other hand, the second article mentioned explores the performance of PSO, GAs and two hybrid algorithms in the search for an optimal design of a profiled corrugated antenna. Finally, similarly to the first article, conclusions are drawn from the experiment, though no suggestions regarding other hybrid possibilities are made, [30, 31].

As mentioned above, article [30] only investigates the performance of PSO and evolutionary optimization in the search for the optimum in four test functions, [30]:

1. The sphere model;
2. The Rosenbrock function;
3. The generalized Rastrigrin function;
4. The generalized Griewank function.

A typical method for initializing the population is to disperse the individuals uniformly across the search space, which is usually symmetric about the origin. Since some of the test functions used in this experiment have optima at or near the origin, the typical method can create some level of bias in the the results. Therefore, besides using the typical method, another initialization method was used, and the comparison between the two allowed to assess the performance of the algorithm and its response to the different initialization schemes used. This second method limited the initial population to a fraction of the search space which was defined as half the distance from the maximum point along each axis back towards the origin, [30].

Besides this, the way the dimensionality affected the performance of the algorithm was also considered. Accordingly, three different dimensions were performed for each initialization method and test function: 10, 20 and 30. That being said, 48 different experiments were undertaken, with a total of 50 trials per experiment, adding up to 2400 total trials. The number of generations was also limited according to the dimension of the experiment and the size of the population was 125 particles for the PSO and 250 individuals, with 250 parents, for the evolutionary optimization. Regarding the evolutionary optimization, tournament selection was the chosen method for selection. Regarding the PSO, the acceleration constant of the particle swarm was set to 2.0 and the velocity was limited between -2 and 2 , [30].

For the case of symmetrical initialization and for the first three functions, PSO found the optima quite quickly and then abruptly flattened out and halted the search process,

while evolutionary optimization appeared more gradual and ultimately outperformed the PSO for all dimensions tested. However, for the fourth function PSO took longer to find the optima but evolutionary optimization maintained the same performance as in the previous three functions, [30].

For the case of asymmetric initialization, the results for the first three functions were quite similar to those obtained for the case of symmetrical initialization, both for PSO and evolutionary optimization. However, evolutionary optimization takes longer to surpass the performance of the PSO, given the asymmetrical nature of the initial population. Besides this, for the fourth function the performance of the PSO improved significantly, suggesting that PSO might have been biased away from the optimum due to the symmetry of the initial population, [30].

Even though differences between the two methods were outlined, they still appear to be very competitive. Although PSO locates the optimum point faster than evolutionary optimization, it cannot dynamically adjust the velocity when approaching it, which dramatically flattens out its performance. In this sense, PSO would largely benefit from a dynamic refinement of the velocity when an optimal region is approached. On the other hand, evolutionary optimization presents a slower approach to the optimum, but outperforms PSO thanks to its capacity to dynamically adjust to the granularity of the local search area. Remarkably, both algorithms seemed to perform consistently despite the initialization method used, [30].

Considering the conclusions drawn, three hybrid approaches were proposed.

- use a self-adaptive technique from the evolutionary optimization to dynamically adjust the velocity in PSO when approaching an optimum
- include a selection method from evolutionary optimization which could reallocate particles performing below average in PSO
- introduce an operator which considers the current best individual of the whole run and biases the creation of offspring in the algorithm of evolutionary optimization

Article, [31], on the other hand, experiments with both PSO and GAs, as well as two hybrid algorithms in the search for an optimal design of a profiled corrugated antenna. The hybrid algorithms used are as follows, [31]:

- GA-PSO hybrid: the GA population is used to start the PSO after 323 fitness evaluations
- PSO-GA hybrid: the PSO population is used to start the GA after 334 fitness evaluations

Regarding the GA, the operators used are mutation, crossover and selection. The fitness function used is the same for all the algorithms tested, allowing for an unbiased

analysis. Regarding the PSO, the movement of the particles contemplates both the particle's personal previous best position as well as the global previous best position. The initialization of the algorithm is done at random, both in terms of position and velocity, and the velocity is limited throughout the whole run. Additionally, each algorithm was given a population of 10 individuals and should stop after completing 600 fitness evaluations, [31].

Since both algorithms employ the majority of their running time evaluating fitness, this limitation had fairly the same impact on the two of them. However, in order to reduce the number of fitness evaluations, the GA did not reevaluate any horn design that had previously been analysed in the run and the PSO did not evaluate particles which flew over the search space, [31].

After conducting the aforementioned experience with the four algorithms, it was noted that the GA improved faster than PSO earlier in the run, but the latter passes GA shortly after. Taking this into account, PSO is said to perform better than GA. As to the hybrid algorithm, PSO outperformed the GA-PSO, however PSO-GA returned the best horn when compared to the other three algorithms. As a final note, PSO is described as a highly competitive and effective method, although PSO-GA seems to be the most promising, [31].

After careful analysis of the two articles, it is possible to understand that the differences in performance between PSO and GA are not necessarily consistent. While the first article indicates GA as the best performing algorithm, the second article states otherwise. Nevertheless, one should take into account the fact that the population used in the latter article was of small dimension, which generally leads to poorer performance of the algorithm, [30, 31].

All in all, hybrid algorithms which combine PSO and GAs seem quite promising according to both articles, but there is still a lot of room for research and development in this up-and-coming area. Furthermore, the introduction of aspects of GAs in the final stages of PSO seems to be the technique which leads to best results, allowing for a refinement of the method near the optimum which guarantees a progressive approach to that same point, [30, 31].

2.10.2 Optimization of ANNs based on EAs

The shortcomings of a neural network are well-known. It can easily get stuck in a local minimum and presents great challenges when it comes to changing the architecture. When used in basic problems, EAs may not be adequate, and their efficiency may not be comparable to that of special algorithms for specific problems. But GAs are robust, nonlinear, and parallel algorithms that can be used in a variety of situations since they are built on prior knowledge of a specific problem. EAs can be used to solve a variety of problems, including connection weight preparation, architecture design, rule learning, input function selection, connection weight initialization, and rule extraction from ANNs. The number

of layers, the number of neurons in each layer, the connecting path between neurons, and other parameters are used to characterize neural network architecture. According to certain performance assessment principles, designing network architecture entails determining the combination of parameters that is best suited to solving a specific problem. It can be difficult to build a network artificially, so ANNs require effective and automated methods of construction, which GA fortunately provides, [7].

2.11 Single-objective (SOO) and multi-objective (MOO) optimization

Optimization usually refers to the process of solving a problem where one or multiple objectives must be minimized or maximized. These objectives may be weighted functions of integer or real variables and they are generally subject to one or more constraints. The objective depends on certain traits or characteristics of the system, which are referred to as variables or unknowns. Solving the optimization problem consists of systematically assigning values to these variables, abiding by the imposed restrictions, and assessing whether the optimal solution has been found - this solution corresponds to the best value of the objective function. Notice that these values must fit within a preset domain. When analysing a real problem, it is common to come across multiple objectives, which might even be conflicting, however real single objective problems can also be found, though they are generally very simplistic, [32, 33].

In order to solve an optimization problem, it is usual to follow a sequence of steps rather than attempt to solve it in one go. These steps are as follows, [32]:

1. check for the need of optimization
2. define the design variables
3. formulate the constraints
4. formulate the objective function/functions
5. choose and optimization algorithm
6. run the algorithm and obtain a solution or set of solutions

Within the field of optimization, two techniques may be used: single variable optimization and multi-variable optimization. Furthermore, for either technique, one can find two optimization scenarios: unrestricted, which means there are no constraints, or restricted, which means there are either equality constraints, inequality constraints or a combination of both, [32, 33].

Single-objective optimization (SOO) encompasses optimization problems which deal with the maximization or minimization of a single objective function based upon

one or more variables subject to any given number of constraints. The objective function may present several local minimum or maximum, and a single global minimum and global maximum, [32, 33].

Within the context of SOO, three different optimization techniques can be found, [32, 33]:

1. Calculus-based or numeric techniques;
2. Enumerative techniques;
3. Guided random search techniques.

The first two technique assumes the existence of derivatives, making it unsuitable to solve the majority of real-life problems. The second technique requires the evaluation of every point in the search space, making them computationally expensive. Evolutionary computation, and thus GAs and PSO, fits in the category of random techniques. This branch is based on enumerative methods, though it is guided to search only promising regions of the search space, making it useful in complex, multidimensional and discontinuous problems with vast search spaces. It encompasses both single point and multiple point search. An example of single point search is simulated annealing, while multiple point search includes evolutionary computation and tabu search, [32, 33].

Multi-objective (multicriteria or multiattribute) optimization (MOO), on the other hand, refers to problems which present two or more conflicting objectives which must be simultaneously optimized with respect to a given set of constraints. If the optimization of one objective coincides with the optimization of the remaining objectives, then the problem under analysis is not considered a MOO problem. In MOO the concept of optimality cannot be directly applied since the answer is generally a set of solutions that define the best trade off between competing objectives. Taking this into account, the quality of a solution is then determined by its dominance and thus the notion of Pareto optimal front is usually employed, [32, 33].

For a given solution to be considered dominant over another solution it must be no worse in all objectives and strictly better in at least one objective. Therefore, it is possible to determine a set of solutions which does not dominate and is not dominated by any potential solution within this set. This set is called a non-dominated solution set and the non-dominated set of the entire feasible decision space is called the Pareto-optimal set. The boundary defined by the set of all points mapped from the Pareto optimal set is called the Pareto optimal front. Taking this concept into account, the main goal in MOO is to find a set of solutions as close as possible to the Pareto-optimal front. Note that the rank of any potential solution in a given set is defined as the number of solutions within that set that dominate it. Therefore, the solutions in the Pareto optimal front have rank 0 as they are not dominated by any other potential solution, [32, 33].

2.11.1 Multi-objective evolutionary algorithms (MOEAs)

Though a panoply of approaches to solve MOO problems can be found in literature, only multi-objective evolutionary algorithms (MOEAs) are mentioned in this text. In the particular case of GAs, where different populations are used for different objectives, a few Pareto-based approaches stand out, namely multiple objective GA (MOGA), non-dominated sorting GA (NSGA), and niched Pareto GA. The aforementioned techniques are generally nonelitist, so it is relevant to mention the most commonly found elitist MOEAs: NSGA-II, the strength Pareto evolutionary algorithm (SPEA), and SPEA2, [33].

It is interesting to note that, though EAs were initially applied to SOO, they can be easily extended to solve MOO problems. That being said, and considering the multitude of different approaches one can adopt to solve MOO problems, an attempt to categorize them is presented below, based on [33]:

Aggregating approaches

1. Weighted sum approach: weights are used to combine multiple objectives. To put it another way, the objective functions are combined into a single function, which is subsequently optimized.
2. Approach based on goal programming: users are requested to set targets or objectives for each objective function, and deviation from these targets is subsequently minimized.
3. Goal attainment-based approach: users must give a target vector and a weight vector that connects the relative accomplishment of the intended goals.
4. ε -Constraint approach: the primary objective function is optimized considering the other objective functions as constraints bounded by some allowable levels ε_i .

Population-based non-Pareto approaches

1. Vector evaluated genetic algorithm (VEGA): this approach makes use of a unique selection operator that creates a number of subpopulations by applying proportional selection to each objective function in turn. This was the earliest MOO attempt using a GA.
2. Lexicographic ordering: the user prioritizes the objectives in order of significance, and the optimization is conducted on these priorities.
3. Game theory-based approach: each objective is seen as a player
4. Use of gender for identifying the objectives: it is permissible to have many parents join to generate a single offspring.

5. Use of the contact theorem: an individual's fitness is characterized by their relative distance from the Pareto front.
6. Use of nongenerational GA: a series of appropriate transformations is used to convert a multi-objective problem to a single-objective problem. Individual fitness is assessed progressively, and in each generation, a single individual is generated through genetic operators, replacing the weakest individual in the population.

Pareto-based non-elitist approaches

1. Multiple objective GA (MOGA): each individual is allotted a rank based on the number of people in the present population by whom it is dominated plus one. Individuals who are non-dominated are ranked first. The fitness of individuals with the same rank is averaged such that they are all sampled at the same rate. The population is then distributed throughout the Pareto-optimal region using a niche generation mechanism. It is important to note that this approach converges slowly, and certain issues with niche size parameters may develop.
2. Niche Pareto GA (NPGA): to decide the winner amongst two candidate solutions, a tournament selection based on Pareto dominance is run on a sample of the population. To assess dominance, a sample of roughly ten individuals is taken, and the non-dominated individual is chosen. If both participants are either dominated or non-dominated, the tournament's outcome is determined by fitness sharing. This approach also has the issue of determining a suitable value for the niche size parameter.
3. non-dominated sorting GA (NSGA): All non-dominated individuals are assigned to a single category, and a dummy fitness value proportionate to population size is assigned. This group is then eliminated, and the remainder of the population is reclassified. The procedure is repeated until all of the individuals in the population have been categorised. In this case, stochastic remainder proportional selection is applied. This approach has a very high convergence rate, however it also has issues with the niche size parameter.

Pareto-based elitist approaches

1. Strength Pareto evolutionary algorithm (SPEA): elitism is expressly manifested by the upkeep of an external population known as an archive. The repository contains all non-dominated solutions from the general population. Each individual in the archive is assigned a strength rating proportionate to the number of individuals it dominates. The fitness of each individual in the present population, on the other hand, is determined by the strength of all the archival non-dominated solutions that dominate it. Furthermore, it uses average linkage clustering to reduce the size of the

archive while increasing diversity in the primary population. This method has been thoroughly tested, and it guarantees that no parameters are necessary for diversity preservation. However, the usage of clustering is seen as a limiting factor.

2. Strength Pareto evolutionary algorithm 2 (SPEA2): to overcome the SPEA's flaws, where individuals dominated by the same archive members have the same fitness values, both the general population and the archive are considered when establishing an individual's fitness values. Furthermore, in order to prevent losing outside and border solutions, which is common when performing the clustering strategy, SPEA2 maintains diversity by applying a density-based strategy on the k th closest neighbor. Unfortunately, this strategy is hampered by computationally costly fitness and density computations.
3. Pareto archived evolutionary strategy (PAES): when producing offspring, a binary representation and bitwise mutation are utilized. The offspring is formed first in this process, and then the goal functions are computed. Following that, its fitness value is compared to that of the parent. Only through dominating the parent can a child be recognized as the next parent. In that instance, the iteration process is resumed. Otherwise, the child is rejected, and the parent produces a new mutant offspring. It should be noted, however, that the parent and offspring may be non-dominant to one other. In such case, the offspring is compared to all members of the archive to see if it outperforms any of them. If it happens, the child is recognized as the new parent, and all dominated solutions are removed from the archive. Otherwise, if the archive does not include any individuals dominated by the child, both the parent and the child are examined for their proximity to the archive's solutions. If the offspring occupies a less populated portion of the search area, it is approved as a parent. In addition, a copy of the offspring is uploaded to the archive.
4. Pareto envelope-based selection algorithm (PESA): the population is divided into two parts: an internal (or primary) population and an external (or secondary) population. PESA uses the same hypergrid search space division as PAES to retain diversity. The hypergrid's selection method is based on its crowding metric. This same metric is used to decide which solutions to provide to the external population, that is the archive of non-dominated individuals discovered throughout the evolutionary process.
5. Pareto envelope-based selection algorithm-II (PESA-II): PESA-II is an improved version of PESA that features region-based selection. In region-based selection, a hyperbox is employed as the selection unit rather than an individual. The selection procedure begins with the selection of a hyperbox (using some conventional selection techniques) and then the selection of a random individual inside that hyperbox. The main advantage of this technique is that it is simple to use and computationally

efficient. However, the gridsize affects the algorithm's outputs, and drawing grids requires previous knowledge of the goal space.

6. Elitist non-dominated sorting GA (NSGA-II): NSGA-II was proposed to overcome the shortcomings of NSGA, particularly its nonelitist design. Individuals in a population are subjected to non-dominated sorting, similar to NSGA, and are awarded rankings as a consequence. A new selection approach called crowded tournament selection is proposed, in which solutions are picked based on crowding distance (which measures the density of a solution's neighborhood and encourages variety maintenance in non-dominated front solutions). To apply elitism, the parent and child populations are joined, and non-dominated individuals from the combined population are handed on to the next generation. Since NSGA-II is one of the most commonly used MOO algorithms, it is covered in greater depth here.

The non-dominated sorting approach is a key feature of the NSGA-II. Given that the non-dominated set of solutions within a set of possible solutions is made up of those that are not dominated by any other solution, the operation of the non-dominated sorting approach is as follows: the non-dominated sorting process first seeks the non-dominated set from the given set of possible solutions. Each of the non-dominated solutions is given a rank of one. The same technique is then performed to the set of potential solutions that does not contain the previously determined set of non-dominated solutions, yielding a new set of non-dominated solutions. Each solution in this new collection of non-dominated solutions is awarded a rating of 2. This method is continued until all of the initial set's solutions have been ranked. As a result, the non-dominated sorting strategy assures that viable alternatives are always prioritized above infeasible options.

All in all, the main steps of the NSGA-II algorithm are as follows:

- Initialize the population.
- While the termination criterion is not met, repeat the following:
 - Evaluate each solution in the population by computing m objective function values.
 - Rank the solutions in the population using non-dominated sorting.
 - Perform selection using the crowding binary tournament selection operator.
 - Perform crossover and mutation (as in conventional GA) to generate the offspring population.
 - Combine the parent and child populations.
 - Replace the parent population by the best members (selected using non-dominated sorting and the crowded comparison operator) of the combined population.
- Output the first non-dominated front of the final population.

Chapter 3

Power transformers

3.1 Fundamentals

Transformers are static devices that use electromagnetic induction to transfer electrical energy from one equipment to another without changing the frequency. This transfer occurs with variations in both voltage and current, but the electrical power remains constant throughout, suffering only small losses. These machines can link circuits with various voltages, which is one of the supporting factors for the widespread use of the alternating current (AC) system for electrical energy transmission and distribution. That being said, transformers enable different components of the power grid, such as generators, transmission lines, distribution networks, and loads, to run at their most suitable voltage levels. They play a significant role in interconnecting the various components of a power grid at different voltage levels, particularly when transmission voltages in certain parts of the system are elevated to higher levels. Transformers are therefore critical connections between the generating stations and the points of consumption. Their integration in a common power grid is represented in figure 3.1, [34, 35, 36].

Note that a transformer is an electromagnetic converter that transforms the electrical energy obtained by its primary winding into magnetic energy, which is then transformed again into electrical energy in other circuits. Therefore, the primary and secondary windings are magnetically coupled rather than electrically connected. Based on whether the secondary voltage is higher or lower than the main voltage, a transformer is called a step-up or a step-down transformer. Bearing that in mind, transformer windings are referred to as high-voltage/low-voltage or low-tension/high-tension windings instead of primary/secondary windings given their ability to either step-up or step-down voltage depending on the need and operation, [34, 35, 36].

3.1.1 Principles and equivalent circuit

As stated above, power remains constant throughout the energy transfer, displaying only small losses due mainly to the resistance presented by the windings. To better

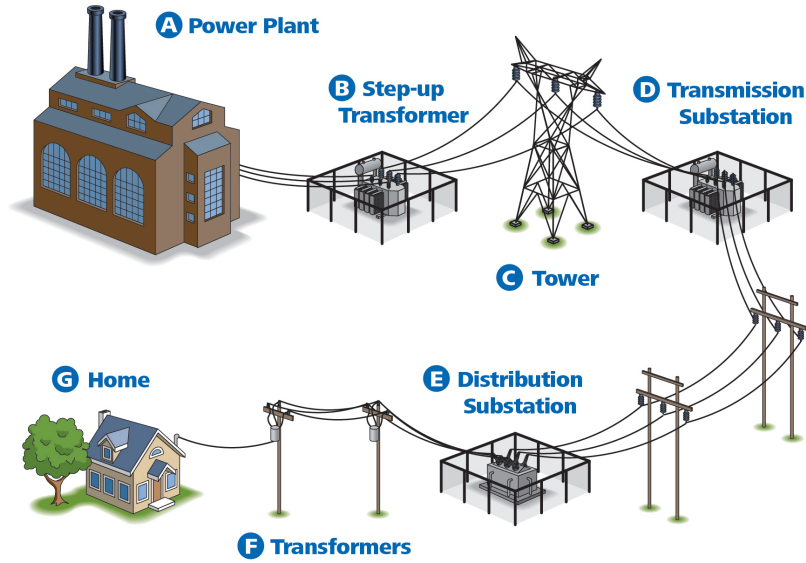


Figure 3.1 The power transmission process, [37].

understand the source of this power loss and how to minimize it, it is relevant to recall the following equations, [34, 35].

$$P = V \cdot I \quad (3.1)$$

where P refers to the power, V refers to the voltage and I refers to the current.

$$R = \rho \cdot \frac{L}{A} \quad (3.2)$$

where R refers to the resistance presented by a given winding, ρ refers to the resistivity of the conductor material, L refers to the length of the winding and A refers to the cross section area of the winding.

Considering equations 3.1 and 3.2, the power loss can be calculated as follows:

$$P_r = I^2 \cdot R = \rho \cdot \frac{L \cdot I^2}{A} \quad (3.3)$$

and the associated voltage drop can be calculated as follows:

$$V_r = I \cdot R = \rho \cdot \frac{L \cdot I}{A} \quad (3.4)$$

Therefore, considering equation 3.1, power loss and voltage drop can be calculated as follows:

$$P_r = \rho \frac{L \cdot P^2}{A \cdot V^2} \quad (3.5)$$

$$V_r = \rho \frac{L \cdot P}{A \cdot V} \quad (3.6)$$

That being said, and considering equations 3.5 and 3.6, it seems that a rise in voltage, V , leads to a plunge in power losses, P_r , and in voltage drop, V_r . Therefore, one can say that electrical power transfer is more efficient when associated with higher voltage values. Alternatively, one could reduce both P_r and V_r by increasing the cross section area of the winding, A . However, an increase in A leads to an increase in the weight of the winding, which in turn leads to greater costs. Hence, it is necessary to consider a trade-off between the minimization of losses and the increased cost associated with larger windings.

3.2 Electric power grid

Electrical energy is carried throughout the power grid at high voltages, however the voltage must be reduced near housing or commercial and industrial facilities. As previously mentioned, this voltage reduction is achieved through a transformer. Additionally, transformers are also used to increase the voltage provided by the generator up to the required value for the transmission of electrical energy. In this way, transformers are crucial in any electric power grid, as they allow for each component of the system to work at its most suitable voltage value, [34, 35].

Figure 3.1 shows a simplification of the electric power grid, mapping out the course from source to final consumer, including generation, transmission and distribution of electric energy. At the beginning there is a power plant (A), where energy is generated, followed by a step-up transformer (B), which elevates the voltage levels to allow for transmission to occur at more efficient levels. Next there is a tower (C) and transmission substation (D). The latter includes a transformer as well, however this one has the purpose of reducing the voltage from transmission level to average tension. Afterwards, another set of transformers reduce the voltage to the lowest value in the distribution substation (E) and transformers (F). Finally, the energy arrives at its destination: the consumer. Here it is represented as a home (G), but it might as well be a commercial or industrial facility, [34, 35].

3.3 Power transformer operation

Transformers operate according to the principle of induction, discovered by Faraday in 1831. This principle states that when a variable magnetic flux interacts with an electrical circuit, it induces an electromotive voltage or force in the circuit. Interestingly, the induced voltage is proportional to the number of turns of the electrical circuit traveled by said flux, [34, 35].

Figure 3.2 displays a representation of a monophasic transformer composed of two windings, with a different number of turns, traversed by the same magnetic flux, ϕ_m . The primary winding has N_1 turns, while the secondary winding has N_2 turns. Therefore, as the number of turns is different in the two circuits, so is the induced force in each of them. The transformer is considered an electromagnetic energy converter since it receives electrical energy in the primary winding and converts it into magnetic energy, reconvertng it back into electrical energy in the second winding, [34, 35].

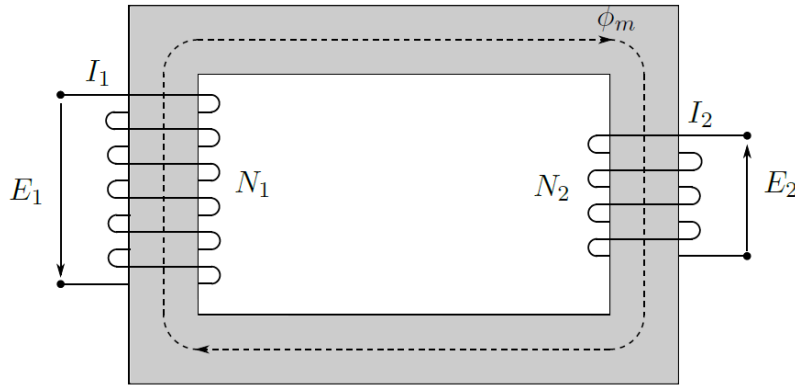


Figure 3.2 Representation of a monophasic transformer composed of two windings, with a different number of turns.

The instantaneous electromotive force induced in the primary and secondary windings due to the magnetic flux, are calculated as follows:

$$E_1 = N_1 \frac{d\phi_m}{dt} \quad (3.7)$$

$$E_2 = N_2 \frac{d\phi_m}{dt} \quad (3.8)$$

Equations 3.7 and 3.8 describe the electromotive forces in the context of the circuit. In the context of the magnetic flux, the induced voltages are symmetrical to the ones stated above.

Disregarding the resistivity of the windings, one can state:

$$V_1 = E_1 \quad (3.9)$$

$$V_2 = E_2 \quad (3.10)$$

$$\frac{E_1}{E_2} = \frac{N_1}{N_2} \quad (3.11)$$

Therefore, taking into account equations 3.1, 3.9, 3.10 and 3.11, the following conclusion may be reached:

$$\frac{E_1}{E_2} = \frac{V_1}{V_2} = \frac{N_1}{N_2} = \frac{I_2}{I_1} \quad \therefore \quad V_1 I_1 = V_2 I_2 \quad (3.12)$$

3.4 Types of transformer

Transformers can be classified according to their role in the power grid or according to their design. The following subsections provide an overview of the types of transformers according to these two types of classification.

3.4.1 On the basis of use: power and distribution transformers

When analysed according to their function, transformers can be divided into power transformers and distribution transformers. The power transformers are located after the generator and are responsible for increasing the voltage up to the transmission value. Besides that, they are also responsible for lowering the voltage - transforming high voltage in medium voltage - near urban or industrial centers. Distribution transformers, on the other hand, are responsible for lowering the voltage - transforming medium voltage into low voltage - so that it can be distributed to consumers, [34, 35].



Figure 3.3 Types of transformer on the basis of use: distribution transformer, Efacec.

3.4.2 On the basis of design: shell and core transformers

Regarding their design, transformers can be classified into core type and shell type. This designation takes into account the type of construction of the active part, which is



Figure 3.4 Types of transformer on the basis of use: power transformer, Efacec.

made up of the magnetic circuit and windings. In the core type transformer the windings are placed around the magnetic circuit. Usually, high voltage and low voltage windings are placed concentrically, with the low voltage winding within the high voltage winding. Note that the windings have a cylindrical shape. In the shell type the magnetic circuit is coiled around the windings. The windings are flat and oval in shape. Furthermore, the low voltage windings are placed side by side with the high voltage windings, [34, 35, 36].

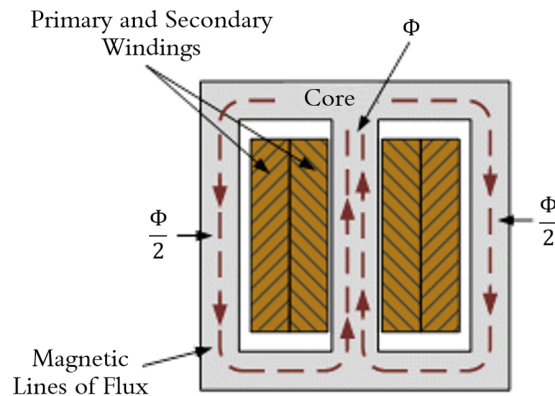


Figure 3.5 Types of transformer on the basis of design: shell transformer.

In order to choose the preferred type of active part of the transformer, and therefore the type of transformer, both function and cost must be taken into account. For example, for distribution transformers it is common to opt for core transformers, since the windings are wrapped around the magnetic circuit in an economical fashion. On the other and, for high power transformers, the shell type is typically used, since greater resistance to short-circuit situations is required. This occurs, for instance, in nuclear power plants. Notice, however, that this type of transformer is more expensive, [34, 35, 36].

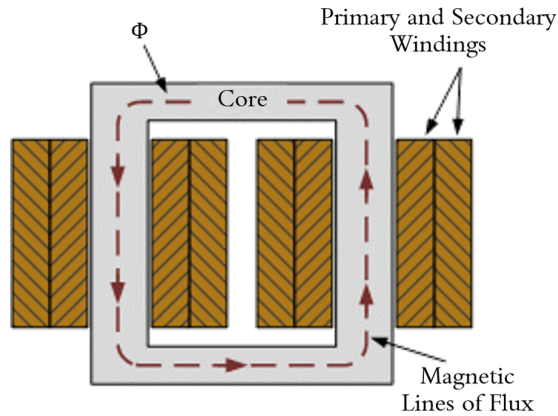


Figure 3.6 Types of transformer on the basis of design: core transformer.

3.5 Structural elements in power transformers

Transformers are machines made out of copper, steel, densified wood, insulating paper and oil. Each of these materials are transformed into various parts, namely the windings, tap changing system, core, tank, and bushings. These elements come into contact with the voltage and current, making up the active component of a transformer, [36].

3.5.1 Windings

A power transformer's windings are its most important component. They are made from copper or aluminum coils that are generally insulated with multiple layers of paper between the turns. As stated above, the core type and shell type windings are the two primary winding designs and technologies that have been evolved over time, displaying many variants. In both cases, the electric basis is the same, but the mechanical design is distinct, [36].

Windings are made with a lot of manual labor and require a lot of experience as well as strict adherence to the highest quality criteria. This is due to the fact that winding conductors are shielded by a form of insulation with minimal mechanical and thermal flexibility, such as varnish or insulating paper. Nonetheless, this insulation form protects the insulation paper from high overvoltages, high overcurrents, short-term overheating, and high mechanical pressures, ensuring that the insulation paper's longevity is not compromised. It should be noted that the transformer's winding insulation cannot be quickly fixed or removed during its service period, and rewinding can only be done in a specialized facility, [36].

3.5.2 Core

The core of a transformer is the most critical and usually the heaviest component. It is made of steel and has a high magnetic permeability, allowing the magnetic flux to pass through it with no resistance. To minimize losses and magnetising current, it is constructed from thin steel sheets with a thickness of a few tenths of a millimetre. The most common method of producing a core is to stack the sheets, which have been cut to the correct size, into automated machines and then manually stack them to form a core, [36].

3.5.3 Tap changer

Additional turns are added to the high voltage windings of most transformers, and some of those turns are related to a system known as a "Tap Changer". During the transformer's service life, it allows for a specific spectrum of voltage variation. There are some movable contacts in the electric circuit of the windings and the tap changer. The De-Energized Tap Changer (DETC) - a mechanically simple type that switches the voltage when the transformer is not loaded - and the On Load Tap Changer (OLTC) - a more complicated type that works while the transformer supplies the load - are the two major forms of tap changers, [36].

3.5.4 Bushings

The bushings connect the windings to a network that runs across the grounded tank. High-voltage bushings are technically challenging, and their failure will quickly result in a transformer explosion. This is because the voltage gradient between the high voltage bushing central component at maximum potential and the grounded tank at a gap of just a few centimetres is one of the largest. The insulating oil under the bushing is highly flammable, and if it sparks, it could produce a lot of steam, open the tank slightly, and then ignite the oil, resulting in an explosion. As a result, the high voltage bushing is designed to tolerate extremely high voltages in a narrow area between the bushing and the transformer tank filled with paper and oil, [36].

3.5.5 Insulating materials

Mineral oil, paper, and pressboard are the three most popular insulating materials used in power transformers. Inside the tank, the mineral insulating oil is weighed in tons and can be used to measure certain important aspects of a transformer's condition as well as any vital incipient faults. The pressboard reinforces the electrical insulation and gives dielectric spacing at various sites, such as in the main duct between the windings, while the paper insulates the winding turns, [36].

Paper, pressboard, and mineral oil, for example, are organic products that deteriorate with time. Solid insulation, unlike other transformer parts and components, is difficult to patch or replace, limiting the transformer's service existence. As a result, the transformer's rigid insulation lifespan translates into the transformer's lifetime, [36].

3.6 Typical loading scenarios for power transformers

The transformer tank is designed to support a set of loads associated with different types of events. The most important events to consider are, [34, 35]:

- Lifting: The lifting eyelets, located at the top of the tank, and the supports, located at the base of the tank, are mechanisms that facilitate lifting the transformer at any given moment. The eyelets are used to lift the transformer with a crane, while the base supports allow lifting, mainly in the same location, without the aid of a crane;
- Transport: Transporting the transformer at the installation location is done with the aid of rollers and transport terminals. The transport terminals are located at the bottom of the tank and the rollers are located under the base plate;
- Seismic and eolian activity: Seismic and eolian activity affect the performance of the tank's connections and support structures;
- Pressure: Pressure occurs when there is an internal issue which leads to the production of a large amount of gas. These gases will cause the internal pressure to rise if the pressure relief mechanism fails to act. This rapid increase may cause the transformer to rupture, which can lead to a fire or oil spillage from inside the tank;
- Short-circuit: Short-circuits can cause large power surges and waves of shock that affect the structure of the core transformer, damaging it and reducing its life span;
- Vacuum: The introduction of vacuum inside the tank is part of the manufacturing process, namely when filling it with oil and removing moisture from its interior. That being said, a relative internal pressure of $0.1MPa$ is created within the tank.

As noted, the tank is subject to a set of loads due to various events, such as pressure, vacuum, elevation, physical activity and transport, which makes its design a complex project. Additionally, there are a number of accessory components that are connected to the tank structure, which make its design even more intricate. Therefore, in an attempt to correctly design a power transformer, one can predict its behavior with the use of numerical methods, such as the FE method. When used correctly, these tools are a cost effective way to facilitate the adequate design of the transformer, [34, 35].

3.6.1 Types of reinforcement

The three types of reinforcements that are typically used are, [34, 35]:

- Beam Reinforcement: this type of reinforcement is used in smaller transformers' tanks since they are more compact and resistant;
- T-shaped reinforcement: this type of reinforcement takes up more space than beam reinforcements, but present less risk of buckling and are more compact than box reinforcement;
- Box reinforcement: this type of reinforcement is used in high power transformers, however it is susceptible to corrosion due to the difficulty of applying anti-corrosion treatments to the inner surface of the box.

Regarding the tank cover plate, which has several accessories and requires a plate with greater thickness, beam or T-shaped reinforcements are generally preferred. On the other hand, the bottom plate of the tank, which supports the total weight of the structure, is usually thicker than the side plates, [34, 35].

The orientation of the reinforcement on the plate can be vertical or horizontal. However, the vertical reinforcement provides a greater contribute to the increase in the resistance of the tank structure. Anyhow, the characteristics of the reinforcements should always abide by space constraints, [34, 35].

With respect to the tank, the design of the reinforcements is a crucial step, since they allow for the minimization of the plate's thickness. Thus, the criterion for introducing reinforcements is based on the minimization of the tank's weight, while guaranteeing it can still withstand all loads applied to it, [34, 35].

Chapter 4

A case study on model updating with synthetic data

In order to achieve the ultimate goal of implementing model updating to power transformers, a simple case study was examined first to assess the quality of some of the methods presented earlier. Consisting of a cube subject to internal pressure, this case study resorts to the FE software *Ansys* to implement and compare various model updating methods, embedded in the aforementioned software.

4.1 Problem specification

The problem at hands consists of a cube ($500 \times 500 \times 500$ mm) with two pipes, one on the side of the cube, to introduce pressure, and another one at the top of the cube, to install a pressure gauge. This cube is made up of S235JR structural steel and the walls are 6 mm thick.

In order to test the optimization tools embedded in *Ansys*, two models were made: a synthetic model (representing the real structure to be experimentally analysed in a real case study) and the nominal model which will be subjected to the FEM updating. The synthetic model is a substitute of the real cube, where the walls are not all exactly 6 mm thick, but instead somewhere between 5.4 mm and 6.6 mm thick. This interval is the dimensional tolerance interval for these elements and the exact measurements used in the synthetic data model are presented in table 4.1. The material considered for the synthetic data model is the S235JR structural steel, which has a Young Modulus of about 210 GPa.

The nominal model, which is the one meant to be subjected to model updating, consists of the same cube with nominal dimensions (6 mm thick walls) and material properties (S235JR structural steel with Young Modulus set to 200 GPa). Notice that the Young Modulus was set to a different value from the one used for the synthetic model in order to investigate the performance of the model updating methodologies when applied to updating this parameter.



Figure 4.1 Cube analysed in the case study.

Table 4.1 Wall thickness for the model used to generate the synthetic data.

Wall	Thickness, t (mm)
Top	6.2
Bottom	5.9
Front	5.7
Back	6.1
Right	5.8
Left	6.3

Notice that, for both models, the cube was considered to be pinned at the corners of its base and all dimensions, except for the thickness of the walls, are the same as the nominal model.

After introducing an internal pressure of 0.1 MPa, the maximum deformation in each wall was evaluated. The analysis of deformation was chosen over strain or tension due to the fact that, in a real experimental analysis, LVDT sensors are more practical when compared to strain gauges and introduce less noise into the acquired signals.

4.2 Ansys: model preparation and synthetic data generation

4.2.1 Preparing the FE model

The aforementioned models were developed using *SpaceClaim*, one of *Ansys* tools. In order to simplify the analysis, all surfaces were converted to *midsurfaces* during geometry preparation (before meshing), allowing the use of shell elements when the mesh for the FE analysis was created. Notice that this approximation is valid since shell elements are used to model structures in which one dimension, in this case the thickness, is significantly smaller than the other dimensions, [38].

After developing the model in *SpaceClaim*, the FE mesh was generated in *Mechanical*, another one of *Ansys* tools. These tools can be accessed through the *Workbench*, which is *Ansys*'s main page. The created mesh is made up of quadrilateral and triangular elements, with greater abundance of the first type. The size of these elements varies according to their location, with a greater density of elements near potentially problematic areas, such as holes or any other area that may be prone to stress concentration.

Afterwards, the boundary conditions and loading scheme were defined. As mentioned before, the four vertices of the base of the cube were deemed restricted and all the welded and bolted connections were considered to be bonded. This is an approximation which minimizes computing time, though it may affect the accuracy of the obtained results. After defining these conditions, an internal pressure of 0.1 MPa was applied to all 6 walls of the cube. These situations are represented in figures 4.4 and 4.5

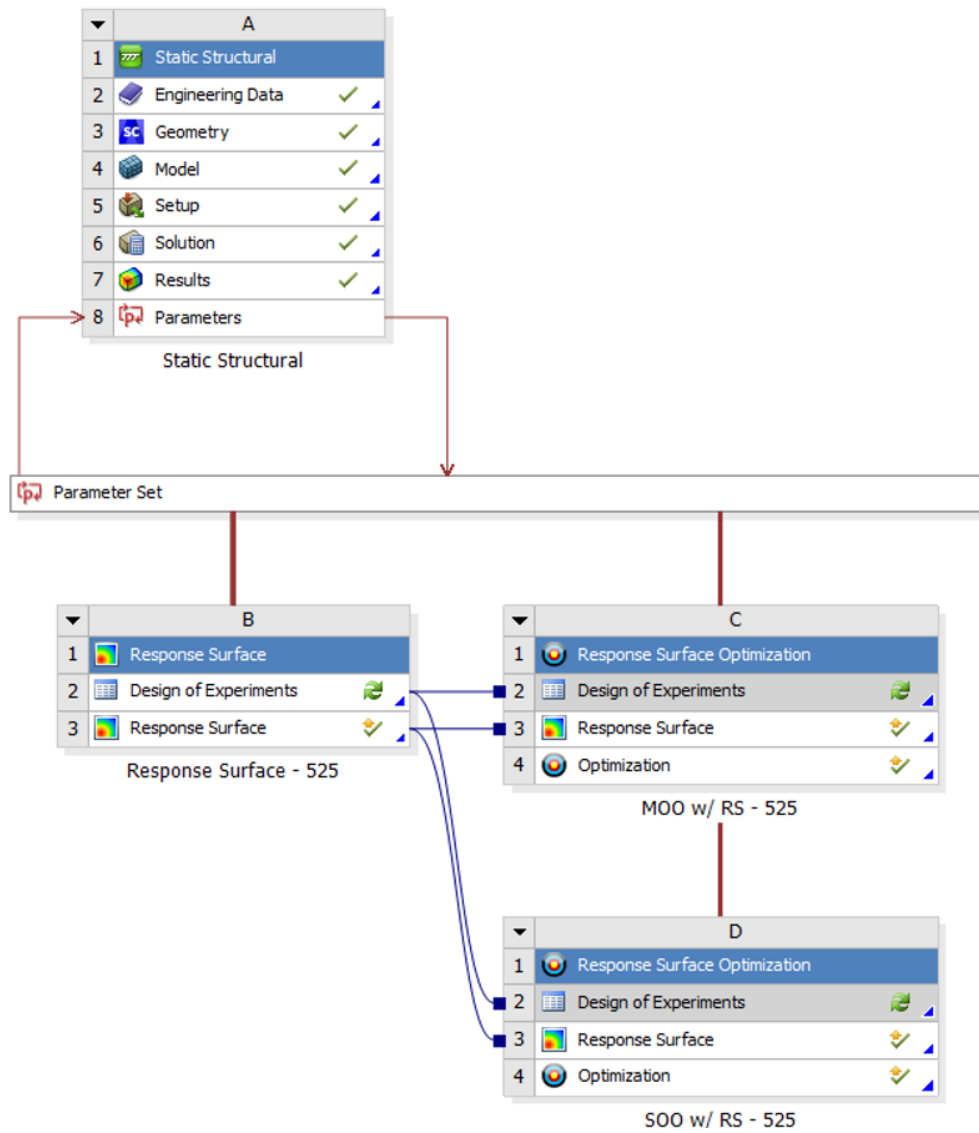


Figure 4.2 *Ansys Workbench*: Static Structural, Parameter Set, Response Surface and Response Surface Optimization blocks.

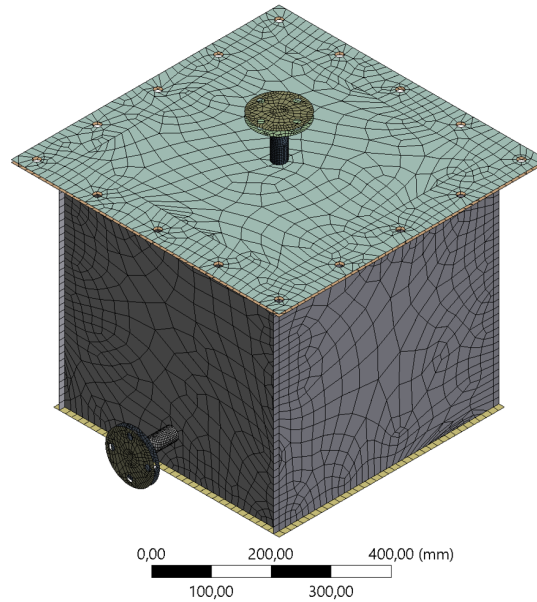


Figure 4.3 FE mesh generated in *Ansys*.

A: Static Structural
Fixed Support

■ Fixed Support

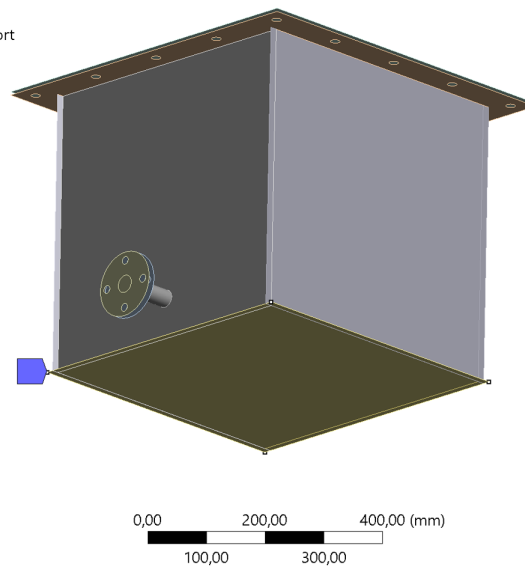


Figure 4.4 Fixed supports applied to the FE model of the cube.

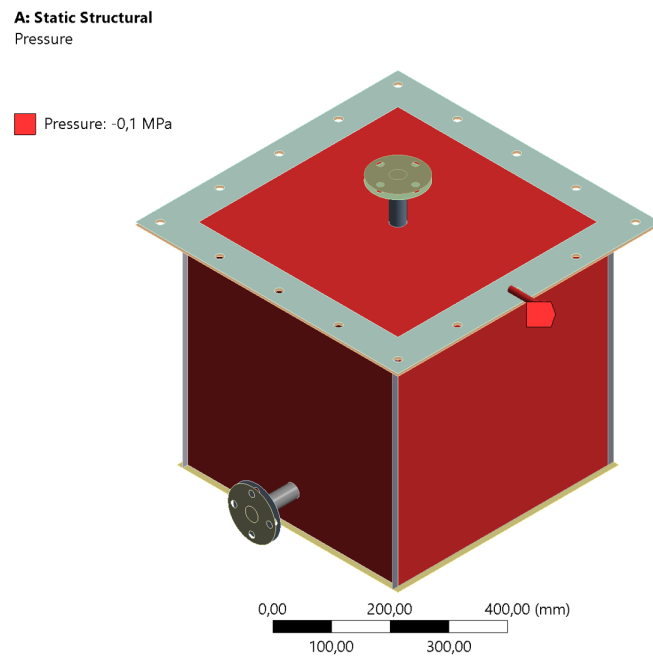


Figure 4.5 Internal pressure applied to the 6 walls of the FE model of the cube.

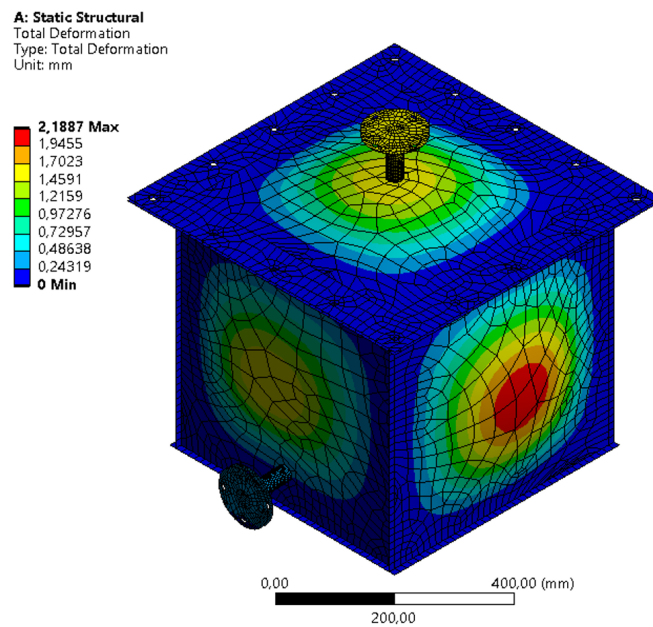


Figure 4.6 Total deformation for the synthetic data calculated in *Ansys*.

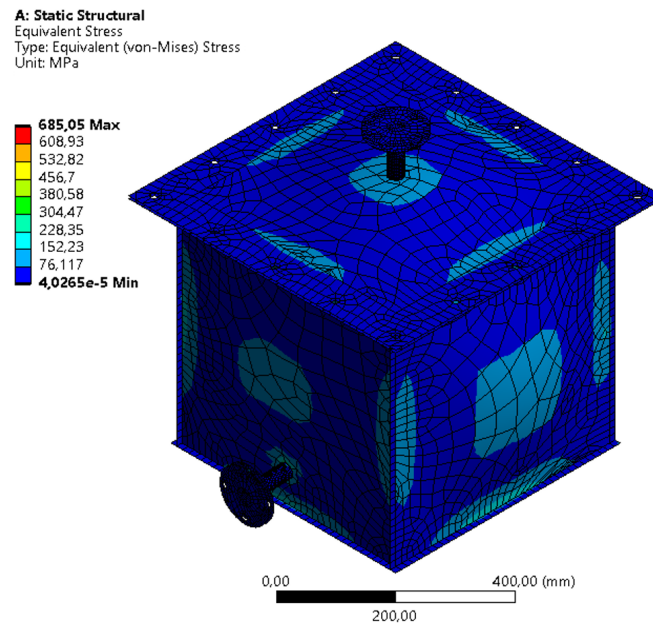


Figure 4.7 Von Mises equivalent stress for the synthetic data calculated in *Ansys*.

Having defined all necessary parameters, it was then possible to run the simulation and retrieve all relevant data. Since the main focus is on the deformation of each wall, the FE model was solved to separately obtain those deformations. Other metrics were also evaluated, however they were not used in the present case study.

4.2.2 Retrieving synthetic data

When evaluating the deformation of each wall, *Ansys* provides a detailed analysis of that surface, displaying its deformation according to a colour scheme. However, only the maximum deformation of each wall was used for the model updating. The maximum deformation occurs at the center of each wall, since the faces of the cube are square and the internal pressure is constant. Notice that, coincidentally, if the experiment were to be conducted on the real cube, the measurement of the deformation would be done at the center of each wall, using LVDT sensors. Therefore, the deformations obtained from the model used to generate the synthetic data are displayed in table 4.2 and refer to the maximum deformation, which occurs at the center of each wall.

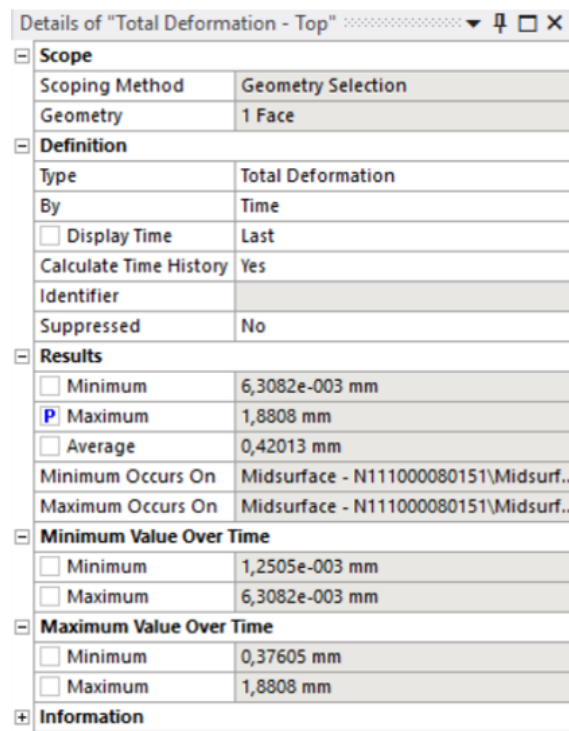
4.2.3 Defining parameters for model updating

The maximum deformation of each wall was defined as a parameter on the model which is meant to be updated (from here on, this model is simply referred to as "model"), as can be seen in figure 4.8. These parameters are used for model updating later on. In the *Parameter Set* tab, found in the *Workbench*, one can go over all the parameters which

Table 4.2 Synthetic data.

Wall	Thickness, t (mm)	Maximum Deformation, δ_{max} (mm)
Top	6.2	1.6935
Bottom	5.9	2.1887
Front	5.7	1.6975
Back	6.1	1.8459
Right	5.8	2.1670
Left	6.3	1.7077

have been defined, as can be seen in figure 4.9. Notice that the thickness of each wall was also set as a parameter, as well as the Young Modulus. These parameters, referred to as 'input parameters', are the ones intended to be updated based on the 'output parameters', that is, the deformation of each wall.

**Figure 4.8** Maximum deformation of the top wall being defined as a parameter.

4.3 Ansys: model updating

As can be seen in figure 4.10, *Ansys* presents various possible optimization approaches within the tab of *Design Exploration*, embedded in the *Workbench*. For the present case study, *Response Surface Optimization* (RSO) and *Direct Optimization* (DO) were implemented separately, with the aim of comparing their performance. It is relevant to note

1	ID	Parameter Name	Value	Unit
2	Input Parameters			
3	Static Structural (A1)			
4	P30	Left Wall\Midsurface 1 Thickness	6	mm
5	P31	Top Wall\Midsurface 1 Thickness	6	mm
6	P32	Bottom Wall\Midsurface 1 Thickness	6	mm
7	P33	Back Wall\Midsurface 1 Thickness	6	mm
8	P34	Front Wall\Midsurface 1 Thickness	6	mm
9	P35	Right Wall\Midsurface 1 Thickness	6	mm
10	P36	Young's Modulus	2E+11	Pa
*	New input parameter	New name	New expression	
12	Output Parameters			
13	Static Structural (A1)			
14	P24	Total Deformation - Top Maximum	1,9446	mm
15	P25	Total Deformation - Bottom Maximum	2,1832	mm
16	P26	Total Deformation - Front Maximum	1,5268	mm
17	P27	Total Deformation - Back Maximum	2,0354	mm
18	P28	Total Deformation - Right Maximum	2,0536	mm
19	P29	Total Deformation - Left Maximum	2,0673	mm
20	P23	SOO	0,27028	mm^2
*	New output parameter		New expression	

Figure 4.9 Parameter set.

that RSO creates a response surface (RS), based on the defined parameters, and then runs an optimization procedure on that RS, while DO runs the optimization procedure directly on the FE model, reevaluating it for each individual or candidate solution. Additionally, the optimization procedure chosen for both scenarios is the same: NSGA-II, a MOGA.

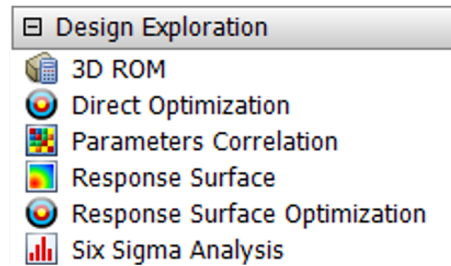


Figure 4.10 Design exploration options available in *Ansys*.

Figure 4.11 presents a scheme of all the conducted experiments. The details of each experiment are explained in the following sub-sections as well as the names used to describe them.

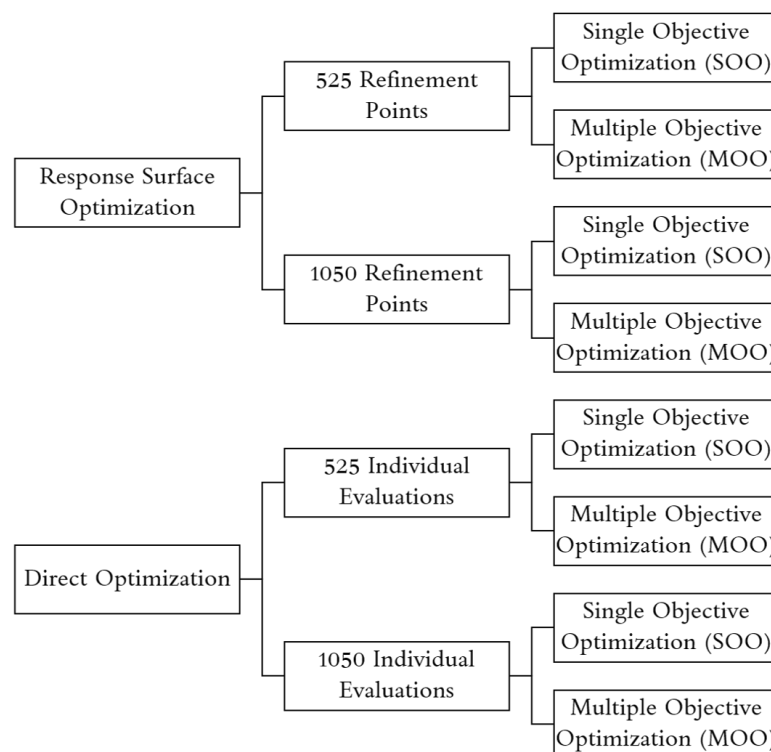


Figure 4.11 Schematic representation of the conducted experiments.

4.3.1 Defining compound output parameters

Notice that, for each of the experiments, two alternative optimization approaches were conducted: SOO and MOO. For the SOO approach, an objective function was created. Although NSGA-II is used in both scenarios, for the MOO various objectives are defined as targets, while for the SOO a single objective function is introduced in the program. For that reason, the objective function must be defined as an output parameter, as it will play a similar role in the SOO procedure as the output parameters (deformation of each wall) play in the MOO procedure. The targets for each of the deformations are presented in table 4.3, and equation 4.1 and figure 4.12 represent the objective function used for the SOO. The objective function is based on the minimum weighted sum method and aims at minimizing the residual between the deformations in the synthetic data and the deformations in the model. For the objective function in the SOO, all deformations are considered to have the same weight, which is in line with the default approach *Ansys* uses in the MOO where no deformation is considered of greater importance than the remaining ones. Recall that the objective for both situations is to update the deformations in the model so that they approximate the values of the deformations provided by the synthetic data.

Table 4.3 Target deformations for each wall based on the synthetic data.

Wall	Target Deformation, δ_t (mm)
Top	1.6935
Bottom	2.1887
Front	1.6975
Back	1.8459
Right	2.1670
Left	1.7077

$$\begin{aligned}
 \text{Objective Function} &= \sum_{n=1}^6 (\delta_n - \delta_{t_n})^2 \\
 &= (\delta_{TopWall} - 1.6935)^2 + (\delta_{BottomWall} - 2.1887)^2 + \\
 &\quad (\delta_{FrontWall} - 1.6975)^2 + (\delta_{BackWall} - 1.8459)^2 + \\
 &\quad (\delta_{RightWall} - 2.1670)^2 + (\delta_{LeftWall} - 1.7077)^2
 \end{aligned} \tag{4.1}$$

4.3.2 Response surface optimization (RSO)

The RSO tool embedded in *Ansys* uses a number of design points to create a polynomial surface, referred to as the RS, which is later on used to conduct the optimization. This RS, which defines the output parameters in terms of the input parameters, can be considered

1	Property	Value
2	General	
3	Expression	$(P24-1,6935[\text{mm}])**2+(P25-2,1887[\text{mm}])**2+(P26-1,6975[\text{mm}])**2+(P27-1,8459[\text{mm}])**2+(P28-2,167[\text{mm}])**2+(P29-1,7077[\text{mm}])**2$
4	Usage	Expression Output
5	Description	
6	Error Message	
7	Expression Type	Derived

Figure 4.12 Definition of the parameter for the SOO objective function in the parameter set.

a meta-model, as it is a simplification of the model under analysis. Note that it is created with the purpose of running an optimization procedure without the need to re-solve the FE model for every evaluation during the optimization method. Naturally, evaluating a given point on this polynomial surface is faster and simpler than conducting a FE analysis, however any flaw or miscalculation in the RS will affect the optimization results.

It is also relevant to note that, if one were to conduct a different type of optimization, aimed at optimizing different parameters, the RS created would not be suitable, and an adequate RS would have to be created. This is one of the drawbacks of creating a meta-model: it only represents certain aspects of the model, which were deemed relevant to a particular analysis, and cannot be used directly for analysis with different aims.

Figure 4.13 presents the *Workbench* blocks referring to the RSO. Notice that the RSO is split into two blocks as it is possible to create a RS and not necessarily run an optimization procedure on it.

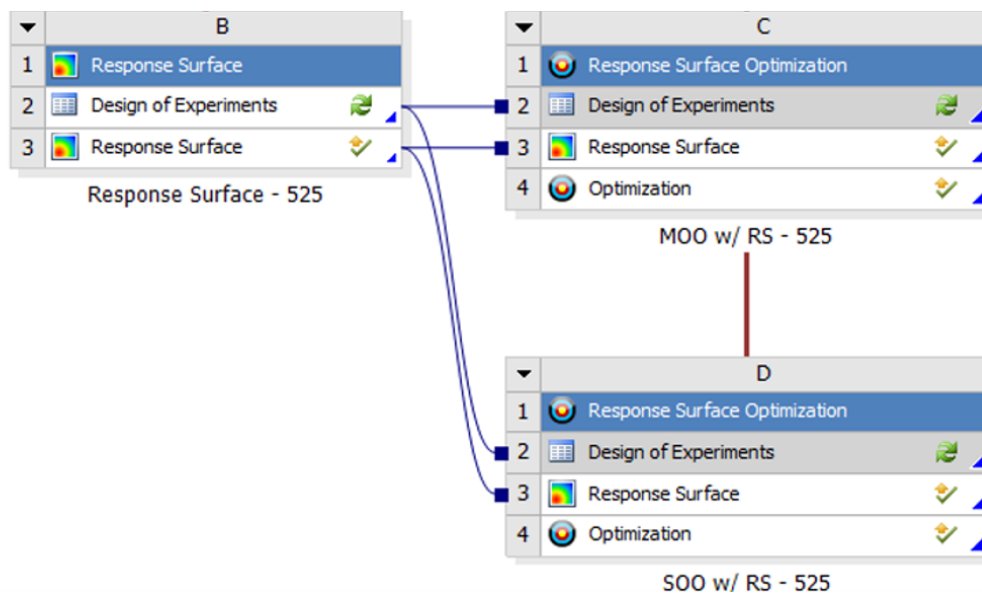


Figure 4.13 RS and RSO blocks in *Ansys Workbench*.

4.3.2.1 Creating the response surfaces (RS)

After defining all parameters, and before creating the RS, one must define the design of experiments (DOE). Here, the lower and upper bounds for each input parameter are defined and the initial design points are created, based on those bounds. The bounds defined for each input parameter were set as is shown in table 4.4 For this analysis, sparse grid initialization was used, which allowed a continuity between the DOE and the method used for creating the RS. Considering that there are 7 input parameters (the thickness of each of the 6 walls and the Young Modulus), each with a lower and upper bound, 15 design points were created for the initial grid, as is presented in figure 4.14. Notice that, since the input and output parameters and respective bounds are the same for both RSs, the initial set of design points is also the same.

Table 4.4 Upper and lower bounds for the input parameters.

Parameter	Unit	Lower Bound	Upper Bound
$t_{TopWall}$	mm	5.4	6.6
$t_{BottomWall}$	mm	5.4	6.6
$t_{FrontWall}$	mm	5.4	6.6
$t_{BackWall}$	mm	5.4	6.6
$t_{RightWall}$	mm	5.4	6.6
$t_{LeftWall}$	mm	5.4	6.6
Young Modulus	GPa	180	220

Name	P8 - Midsurface - N111000080120 (Midsurface1.Thickness (mm))	P15 - Midsurface - N111000080130 (Midsurface1.Thickness (mm))	P16 - Midsurface - N111000080151 (Midsurface1.Thickness (mm))	P17 - Midsurface - N111000080160 (Midsurface1.Thickness (mm))	P18 - Midsurface - N111000080111 (Midsurface1.Thickness (mm))	P19 - Midsurface - N111000080130 (Midsurface1.Thickness (mm))	P22 - Young's Modulus (Pa)
1 DP	6	6	6	6	6	6	2E+11
2	5,4	6	6	6	6	6	2E+11
3	6,6	6	6	6	6	6	2E+11
4	6	5,4	6	6	6	6	2E+11
5	6	6,6	6	6	6	6	2E+11
6	6	6	5,4	6	6	6	2E+11
7	6	6	6,6	6	6	6	2E+11
8	6	6	6	5,4	6	6	2E+11
9	6	6	6	6,6	6	6	2E+11
10	6	6	6	6	5,4	6	2E+11
11	6	6	6	6	6,6	6	2E+11
12	6	6	6	6	6	5,4	2E+11
13	6	6	6	6	6	6,6	2E+11
14	6	6	6	6	6	6	1,8E+11
15	6	6	6	6	6	6	2,2E+11

Figure 4.14 Initial design points created in the DOE for both RSs.

In order to create the RS, *Ansys* offer various methods, such as, [39]:

- **Genetic Aggregation** - uses a GA to produce populations of alternative RSs computed in parallel in order to pick the optimal RS. Each RS's fitness function is utilized to determine which one offers the best approach. It considers the accuracy of the RS on the design points as well as the stability of the RS (crossvalidation).
- **Full 2nd-Order Polynomials** - performs a regression analysis which is a statistical approach that makes use of the connection between two or more quantitative variables in order to estimate one dependent variable from the others. A regression analysis implies that there are n sampling points and that the corresponding values of the output parameters are known for each sample point. Based on these sample

points, the regression analysis reveals the connection between the input parameters and the output parameter. This connection is also affected by the regression model used. For the regression model, a second-order polynomial is usually preferred. This model is typically an approximation of the real input-to-output relationship, and it only produces a true and precise relationship in few instances. The RS is the resultant estimate of the output parameter as a function of the input variables after this connection is identified.

- **Kriging** - is a meta-modeling method that improves response quality and accommodates higher order fluctuations of the output parameter. It is a precise multi-dimensional interpolation that combines a polynomial model similar to that of the conventional RS — which gives a "global" model of the design space — with local deviations such that the Kriging model interpolates the Design of Experiments (DOE) points. Kriging allows for the refining of continuous input parameters, including those with manufacturable values. Discrete parameters are not supported. Kriging's success is dependent on its internal error estimator's capacity to enhance response surface quality by creating refinement points and adding them to the parts of the RS that require improvement the most.
- **Non-Parametric Regression (NPR)** - is designed for outputs with consistently strong nonlinearity with respect to inputs. NPR is a method that belongs to the Support Vector Method (SVM) family. These are data categorization algorithms that divide data groups using hyperplanes. The regression technique operates in the same way. The key distinction is that the hyperplane is utilized to classify a subset of the input sample vectors that are judged enough to represent the output in question. The support vector set is the name given to this subset. The RS's internal parameters are set to constant values and are not optimized. The values are obtained through a series of benchmark tests, and they achieve a balance between RS accuracy and computing speed. The present settings produce decent outcomes for a wide range of issues. However, significant oscillations between the DOE points may be seen for some problem types (such as those dominated by flat surfaces or lower order polynomials).
- **Neural Network** - is a mathematical approach based on the human brain's natural neural network. To interpolate a function, a network with three layers (input, hidden, and output) is constructed with weighted connections between them. To create the approximation, this approach employs a small number of design points. It works best when the number of design points and intermediate cells is large. It can provide fascinating outcomes with a variety of settings.
- **Sparse Grid** - allows for the refining of continuous parameters, including those with manufacturable values. Discrete parameters are not supported. Sparse Grid employs

an adaptive RS, which implies it automatically refines itself. A dimension-adaptive algorithm determines which dimensions are most essential to the goal functions, minimizing computing cost. The refining criterion are customizable. Sparse Grid utilizes an automatic local refinement method to find the parts of the RS that require the greatest refining when updating it. It then focuses refinement points in these regions, allowing the RS to attain the desired degree of accuracy faster and with fewer design points.

For this particular analysis, the Sparse Grid method was used. This method is particularly interesting in terms of time efficiency, which makes it attractive for the present study, as the purpose is to compare DO, which is time consuming, with a more expeditious method, such as the RSO, and evaluate their competitive features. Sparse grid uses an adaptive RS, which means that it refines itself automatically. This method starts out by creating a coarse mesh which is refined based on validation points until a criterion is met. The termination criteria are as follows:

- Level of accuracy
- Maximum number of refinement points
- Maximum depth

The maximum depth is the maximum number of times a mesh can be refined in one direction. Once the maximum depth for a direction is reached, there is no further refinement in that direction. For this particular case, the initial mesh is made up of 16 points and 32 validation points were considered. The maximum depth was set to 10 and the maximum relative error was set to 0.1%. In order to assess the adequate number of refinement points to be used, two RSs were created. The maximum number of refinement points was set to 525 for one of them and 1050 for the other one. This information is presented in tables 4.5 and 4.6

Table 4.5 Parameters for generating a RS with 525 design points.

RS with 525 Design Points	
Initial number of design points	15
Number of validation points	32
Maximum Relative Error (%)	0.1
Maximum Depth	10
Maximum Number of Refinement Points	525

After creating both RSs, the evolution of the error was analysed. Table 4.7 displays the relative error for each of the RSs and figures 4.15 and 4.16 display graphs which present the relative error as a function of the number of refinement points, providing an insight into the evolution of the RS. Figure 4.17 represents a comparison of the maximum relative

Table 4.6 Parameters for generating a RS with 1050 design points.

RS with 1050 Design Points	
Initial number of design points	15
Number of validation points	32
Maximum Relative Error (%)	0.1
Maximum Depth	10
Maximum Number of Refinement Points	1050

error as a function of the number of refinement points for the RSs with 525 and 1050 design points

Table 4.7 Relative error for each of the RSs.

Response Surface	Relative Error (%)
525 Design Points	0.4985
1050 Design Points	0.1531

Since the relative error is greater than 0.1% for both RSs, one can state that they did not converge. However, the relative error for both situations is considerably low (bellow 0.5%), making a good case for the quality of the RS.

After analysing the evolution of the relative error for both RSs, it seems that after about 400 design points there is no significant improvement to the quality of the RS, evaluated according to the relative error. To better understand this trend, table 4.8 presents the relative error for various numbers of design points for each of the RSs.

Table 4.8 Evolution of the maximum relative error for the RS with 525 design points and for the RS with 1050 design points.

No. of Refinement Points	Maximum Relative Error (%)	
	RS with 525 Design Points	RS with 1050 Design Points
100	4.910	4.910
200	3.942	3.942
300	2.279	2.279
400	1.279	1.279
500	0.505	0.505
600	-	0.388
700	-	0.306
800	-	0.2221
900	-	0.176
1000	-	0.155

Table 4.8 presents relative errors inferior to 5%, even for a number of design points as low as 100, which is an acceptable value. Therefore, even if the RS was created using only

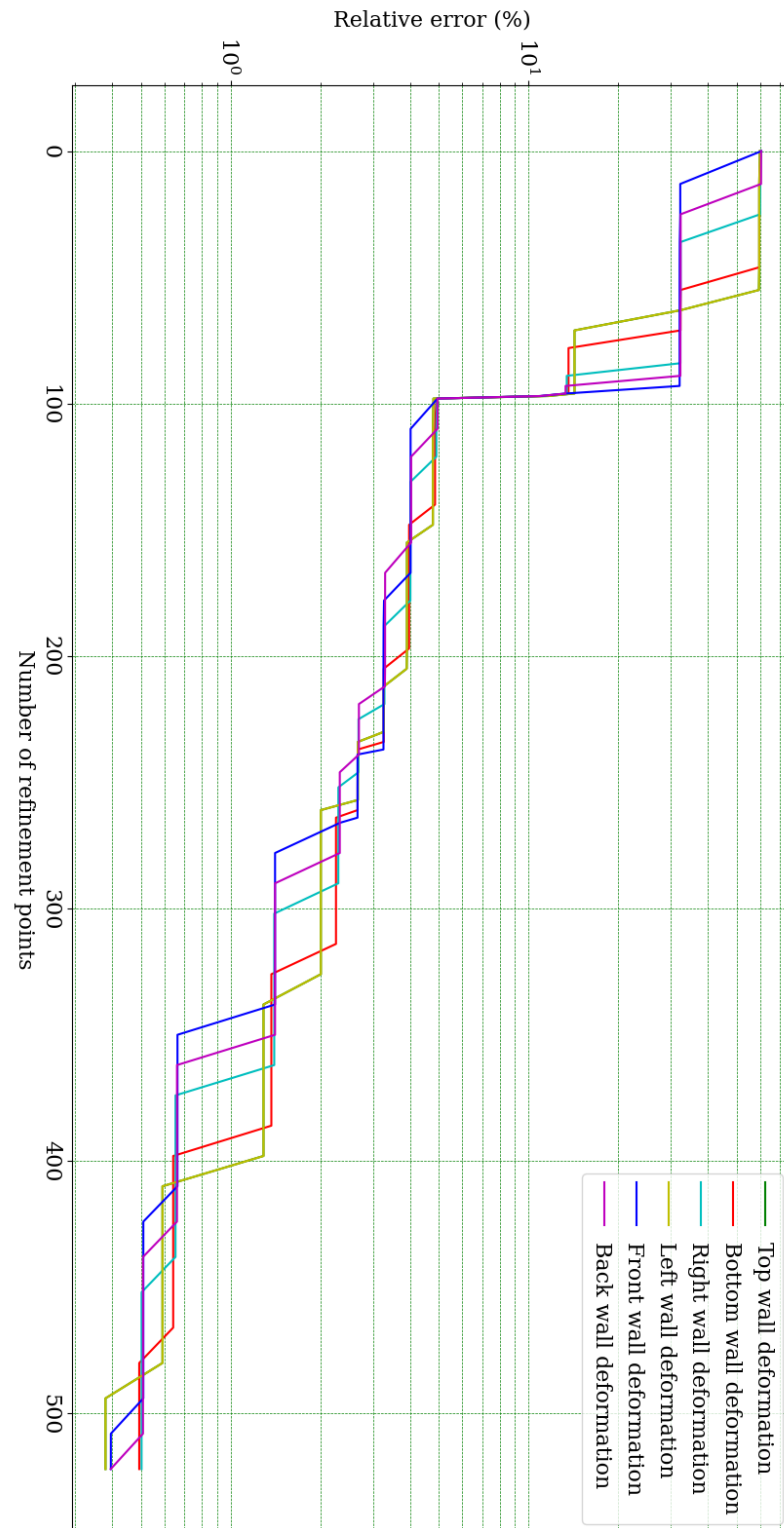


Figure 4.15 Relative error as a function of the number of refinement points for the RS with 525 design points.

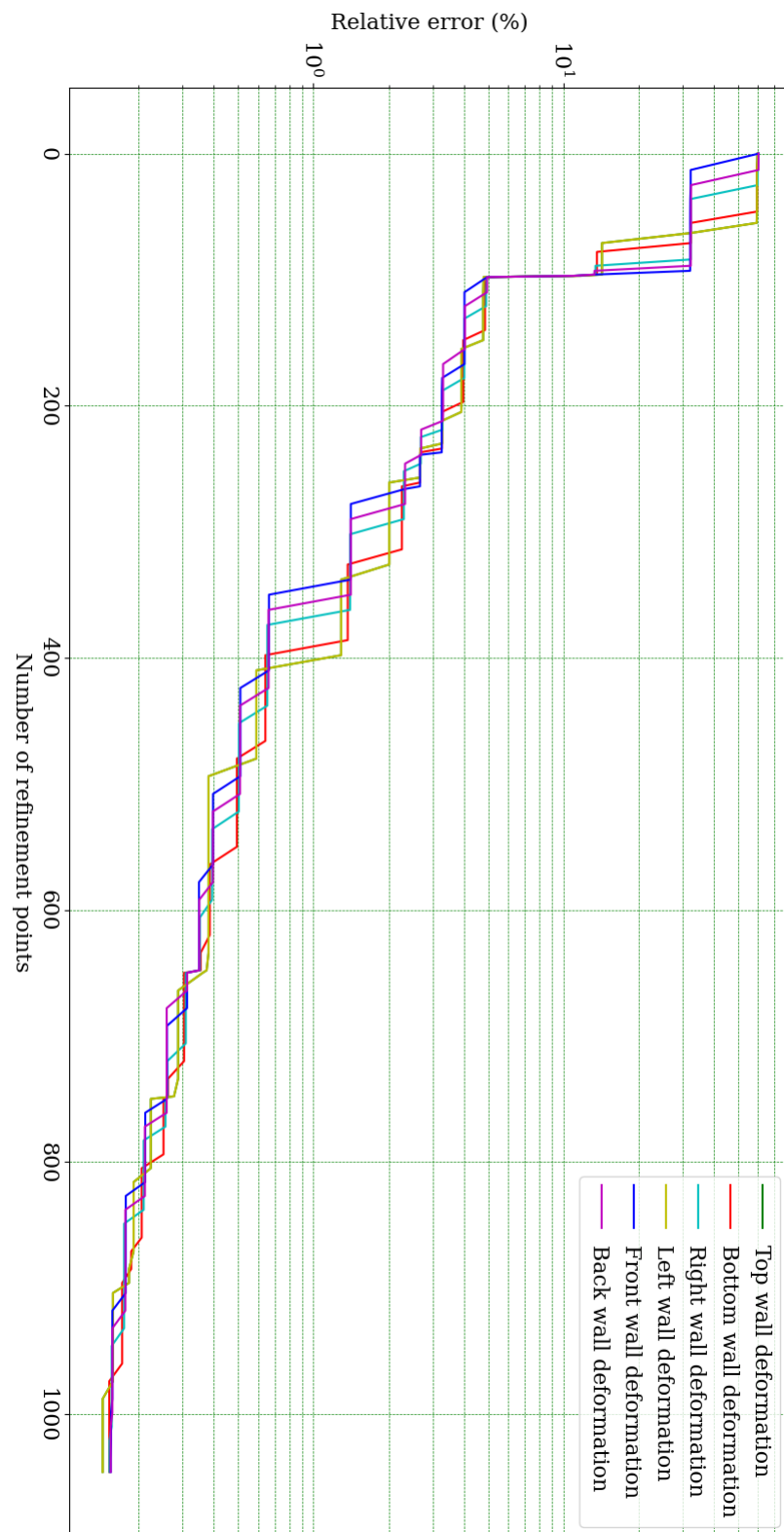


Figure 4.16 Relative error as a function of the number of refinement points for the RS with 1050 design points.

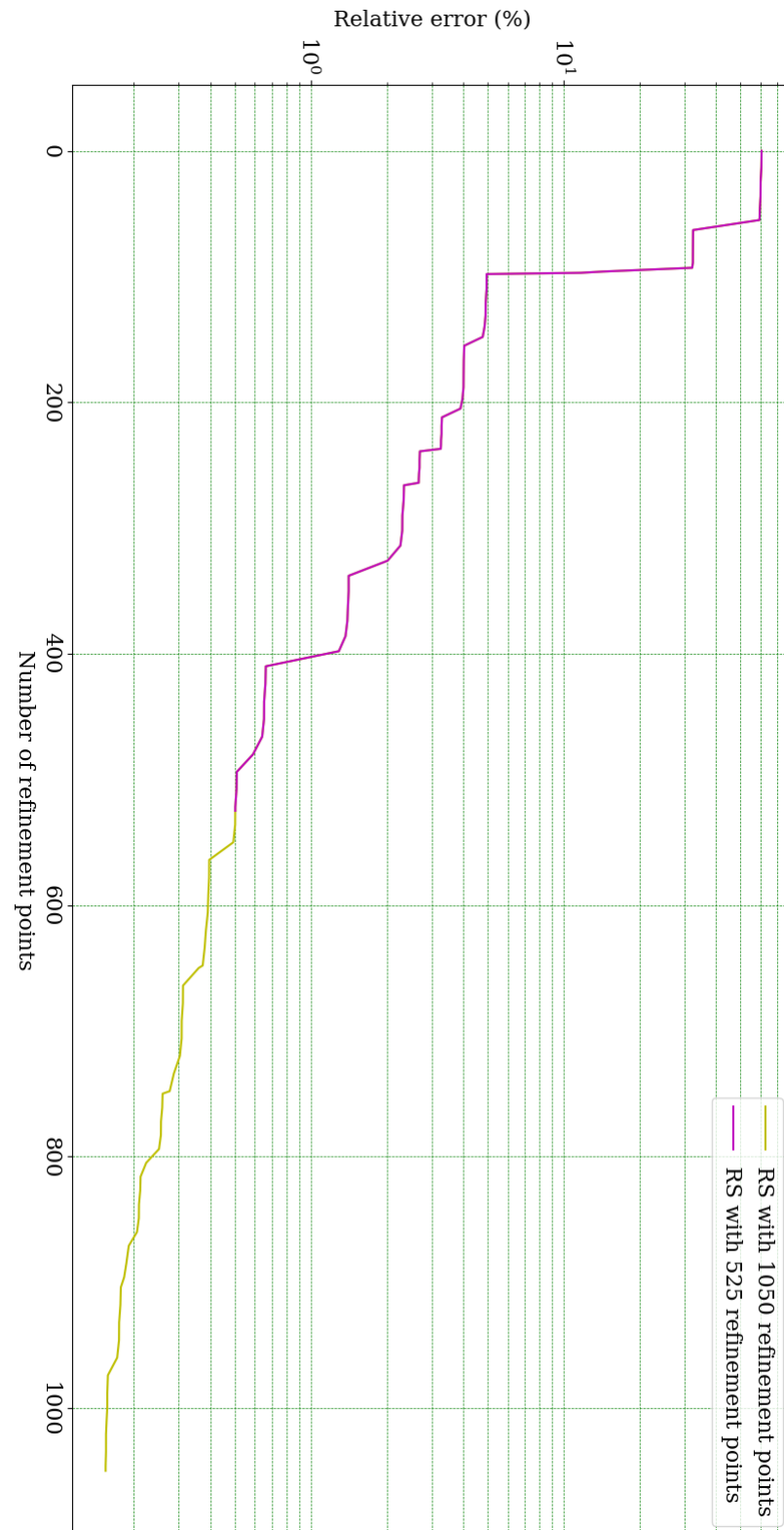


Figure 4.17 Comparison of the maximum relative error as a function of the number of refinement points for the RSs with 525 and 1050 design points.

100 design points, it would still present satisfactory quality. However, one should bear in mind the simplicity of the present problem. If more parameters were used, specially parameters such as strains and stresses, the RS might not present an equivalent level of quality as these parameters are not obtained directly.

Additionally, since the creation of the RS follows a deterministic method, it is not necessary to reevaluate and compare the results obtained by creating new RSs using the same design parameters. This characteristic (determinism) is represented in table 4.8 and in figure 4.17 where one can see that the evolution of the maximum relative error is unaffected by the number of refinement points used. Therefore, the evolution of the maximum error for both RSs (using 525 or 1050 refinement points) is coincident.

4.3.2.2 Running the optimization procedures

After generating the RSs, it was possible to conduct the optimization procedures using NSGA-II (Non-dominated Sorted Genetic Algorithm-II), a MOGA embedded in *Ansys*. As previously mentioned, this is a MOGA based on controlled elitism concepts which employs both crossover and mutation operators. As expected, its criteria can be tailored to better fit the problem at hands.

The criteria for the NSGA-II include:

- Estimated number of design points
- Number of initial samples
- Number of samples per iteration
- Maximum allowable Pareto percentage
- Convergence stability percentage
- Maximum number of iterations
- Maximum number of candidates

The estimated number of design points refers to the number of initial samples combined with the number of samples per iteration multiplied by the maximum number of iterations.

$$n_{design\ points} = n_{initial\ samples} + n_{samples\ per\ iteration} \times n_{iterations} \quad (4.2)$$

The maximum allowable Pareto percentage refers to the maximum desired ratio between the number of Pareto points and the number of samples per iteration, which is a metric used to control selection pressure and therefore avoid premature convergence. When this percentage is reached, the optimization is considered to have converged. However, the optimization stops before the maximum allowable Pareto percentage is met if the maximum number of iterations is reached. The convergence stability percentage represents

the stability of the population based on its mean and standard deviation. This criterion allows for the minimization of the number of iterations performed while still reaching the desired level of stability. When the specified percentage is reached, the optimization is considered to have converged. The maximum number of candidates simply refers to the maximum amount of potential solutions the software should provide at the end of the optimization process, [39].

At this point, two optimization procedures were conducted for each of the RSs: one using a MOO procedure and another one using a SOO procedure. For each of these, the RSO criteria must be defined. There, the output parameters which should be optimized are defined, as well as the targets they must meet. Bearing in mind the targets for each of the deformations, presented before in table 4.3, the MOO deformation targets were defined as can be seen in figure 4.18. For the SOO, figure 4.19 represents the objective function (parameter P23) being defined as the output to be optimized. Contrary to what happened for the MOO, here the goal is simply to minimize that parameter, thus no target is defined.

	A	B	C	D	E
1	Name	Parameter	Objective		
2			Type	Target	Tolerance
3	Seek P24 = 1,6935 mm	P24 - Total Deformation - Top Maximum	Seek Target	1,6935	0,0001
4	Seek P25 = 2,1887 mm	P25 - Total Deformation - Bottom Maximum	Seek Target	2,1887	0,0001
5	Seek P26 = 1,6975 mm	P26 - Total Deformation - Front Maximum	Seek Target	1,6975	0,0001
6	Seek P27 = 1,8459 mm	P27 - Total Deformation - Back Maximum	Seek Target	1,8459	0,0001
7	Seek P28 = 2,167 mm	P28 - Total Deformation - Right Maximum	Seek Target	2,167	0,0001
8	Seek P29 = 1,7077 mm	P29 - Total Deformation - Left Maximum	Seek Target	1,7077	0,0001

Figure 4.18 Objective definition for MOO : deformation targets.

1	Name	Parameter	Objective		
2			Type	Target	Tolerance
3	Minimize P23	P23 - SOO	Minimize	0	

Figure 4.19 Objective definition for SOO: minimization of the objective function defined under parameter P23.

The criteria used for both SOO and MOO is presented in table 4.9. Both approaches were defined using the default criteria to facilitate comparing their performance, however the convergence stability percentage was set to 0. Since it is almost impossible to reach said value, this approach is equivalent to eliminating that termination criteria. In fact, assuming that the maximum Pareto percentage isn't reached, this allows the simulation to run all possible iterations. Additionally, notice that the estimate number of design points (evaluated through the RS) is very large, which should lead to a more accurate outcome. Therefore, any deviation in the obtained results must not be traced back to the quality

of the optimization but instead to the quality of the RS. This is relevant to the present study since the goal is to obtain reliable results in the optimization procedure, allowing for an unbiased comparison between RSO and DO.

Table 4.9 Criteria used for the optimization procedure in all of the RSO experiments.

Criteria	Value
Estimated number of design points	33600
Initial samples	7000
Samples per iteration	1400
Maximum Pareto percentage	70
Convergence stability percentage	0
Maximum number of iterations	20

4.3.2.3 Results and comments

The performance of all the optimization approaches is presented in tables 4.10 to 4.16. Notice that, for MOO, various solutions are presented. This is due to the fact that MOO does not lead to a single solution but instead an array of solutions which are ranked according to the concept of dominance, as mentioned above. Therefore, each solution must be analysed to better understand its performance for each of the objectives. To facilitate that analysis, the error for the updated thickness of each wall (equation 4.3) was calculated for each of the candidate solutions, as well as the sum of the absolute value of these errors (combined absolute error - CAE (%) - equation 4.4). Additionally, the absolute value of the error of the updated Young Modulus was also taken into account for another combined absolute error (CAEYM - equation 4.5). Note that this combined absolute error includes all thicknesses and the Young Modulus. For SOO, on the other hand, one single solution is presented here as there is a single objective to be fulfilled. The aforementioned CAE and CAEYM were also calculated for this solution. Notice, however, that *Ansys* still provides various candidate solutions for the SOO, as it also applies the NSGA-II to the SOO problem. Nonetheless, the concept of dominance does not suit SOO, therefore only the best candidate is analysed here, as the remaining candidates are always worse.

$$Error (\%) = \frac{t_{UpdatedModel} - t_{SyntheticData}}{t_{SyntheticData}} \times 100 \quad (4.3)$$

$$Combined\ Absolute\ Error\ (\%) = CAE\ (\%)$$

$$= \sum_{n=1}^6 \left| \frac{t_{UpdatedModel,n} - t_{SyntheticData,n}}{t_{SyntheticData,n}} \times 100 \right| \quad (4.4)$$

$$\begin{aligned}
\text{CAE}_{w/\text{YoungModulus}} (\%) &= \text{CAEYM} (\%) \\
&= \text{CAE} (\%) + \frac{\text{E}_{\text{UpdatedModel}} - \text{E}_{\text{SyntheticData}}}{\text{E}_{\text{SyntheticData}}} \times 100
\end{aligned} \tag{4.5}$$

Besides that, the sum of the squared differences between the deformation in the updated model and the deformation in the synthetic data was also calculated for the MOO. This is equivalent to the value of the SOO objective function, which is also presented here, allowing for direct comparison between the two. This information is presented in tables 4.14 to 4.16, where:

- δ_{max} refers to the maximum deformation of a given wall in the updated model
- *Sq. Diff.* refers to the squared difference between the maximum deformation in a given wall for the updated model and the synthetic data (equation 4.6)
- *Sum Sq. Diff.* refers to the sum of the squared differences (*Sq. Diff.*) of all the walls for a given candidate solution (equation 4.7). Notice that, for the case of SOO, this value corresponds to the value of the objective function and is therefore automatically calculated by *Ansys*

$$\text{Sq. Diff.} = (\delta_{\text{UpdatedModel}} - \delta_{\text{SyntheticData}})^2 \tag{4.6}$$

$$\text{Sum Sq. Diff.} = \sum_{n=1}^6 (\delta_{\text{UpdatedModel},n} - \delta_{\text{SyntheticData},n})^2 \tag{4.7}$$

Table 4.10 Details of the SOO and MOO experiments.

Type of Optimization	SOO		MOO	
Number of Refinement Points	525	1050	525	1050
No. of iterations:	20	20	20	20
Sample size:	1400	1400	1400	1400
No. of evaluations:	32893	32893	32893	32893
Stability percentage:	0.0601	0.0684	0.3530	0.2936
Pareto percentage:	0.0714	0.0714	0.0714	0.0714

Although both optimization approaches were defined using the same criteria, SOO leads to better results across all evaluated metrics (CAE, CAEYM and sum of the squared differences). In fact, the sum of the squared differences suggests that the SOO approach is superior to the MOO approach, presenting results around ten times better for both types of RS. Concerning the CAE and CAEYM, they are lower for SOO across all experiments using RSO, presenting results at least 0.5% better than those obtained using MOO. Interestingly,

Table 4.11 Results from the SOO experiments using the RSs with 525 design points and 1050 design points.

Single Objective Optimization					
Number of Design Points		525		1050	
Parameter	Unit	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2199	-0.32	6.2393	-0.63
$t_{BottomWall}$	mm	5.9143	-0.24	5.9345	-0.59
$t_{FrontWall}$	mm	5.7200	-0.35	5.7403	-0.71
$t_{BackWall}$	mm	6.1155	-0.25	6.1291	-0.48
$t_{RightWall}$	mm	5.8104	-0.18	5.8350	-0.60
$t_{LeftWall}$	mm	6.3115	-0.18	6.3450	-0.71
CAE	%	-	1.53	-	3.72
Young Modulus	GPa	208.6	0.59	206.7	1.50
CAEYM	%	-	2.12	-	5.22

however, candidate solution number 3, for the MOO using the RS with 525 design points, seems to perform better according to these metrics than SOO using the same RS. It also performs better than the remaining 5 candidates according to those two metrics. This solution, on the other hand, does not perform as well in terms of sum of the squared differences. For the case of MOO using the RS with 1050 design points, candidate solution number 5 seems to outperform the remaining 5 when evaluated according to the CAE and CAEYM. Anyhow, SOO still outperforms them, specially in terms of the CAEYM.

Additionally, it is relevant to note that, regarding the sum of the squared differences, SOO seems to worsen, while MOO does not seem to improve significantly with an increase in the number of design points used for generating the RS. This may mean that the RS using 525 design points reflects the model it is supposed to model with sufficient accuracy and the worse results found using a RS with a higher number of refinement points may be due to the nature of the optimization approach. Bearing that in mind, increasing the number of design points beyond that value may not be necessary. Interestingly, the CAE and CAEYM seem to worsen with the increase of the number of design points, for both SOO and MOO. Therefore, it is necessary to address whether this unexpected error derives from the RSs or the optimization procedures. One simple way to analyse whether the RSs accurately reflect the model is by evaluating the candidate solutions obtained from the RSO in the FE model and comparing the results obtained.

In order to present a more accurate description of the evolution of the quality of the RSs (and the results obtained from the RSO) according to the number of design points used, two additional RSs was created, using 125 and 725 design points. For these RSs and corresponding optimization procedures, the aforementioned metrics (CAE, CAEYM and sum of the squared differences) were also calculated and the obtained results can be

Table 4.12 Results from the MOO experiment using the RS with 525 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2253	-0.41	6.3323	-2.13	6.2007	-0.01
$t_{BottomWall}$	mm	5.9301	-0.51	5.9888	-1.51	5.8949	0.09
$t_{FrontWall}$	mm	5.7277	-0.49	5.7977	-1.71	5.7045	-0.08
$t_{BackWall}$	mm	6.1590	-0.97	6.1195	-0.32	6.1345	-0.56
$t_{RightWall}$	mm	5.8297	-0.51	5.8922	-1.59	5.7955	0.08
$t_{LeftWall}$	mm	6.3210	-0.33	6.3332	-0.53	6.3283	-0.45
CAE	%	-	3.22	-	7.79	-	1.27
Young Modulus	GPa	207.1	1.34	200.9	4.28	210.7	-0.36
CAEYM	%	-	4.55	-	12.07	-	1.63

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2253	-0.41	6.1754	0.40	6.2612	-0.99
$t_{BottomWall}$	mm	5.9301	-0.51	5.8949	0.09	6.0049	-1.78
$t_{FrontWall}$	mm	5.7390	-0.68	5.7036	-0.06	5.8040	-1.82
$t_{BackWall}$	mm	6.1634	-1.04	6.1345	-0.56	6.1930	-1.53
$t_{RightWall}$	mm	5.8339	-0.59	5.7955	0.08	5.9053	-1.82
$t_{LeftWall}$	mm	6.3274	-0.44	6.2575	0.67	6.4375	-2.18
CAE	%	-	3.66	-	1.87	-	10.11
Young Modulus	GPa	207.1	1.34	210.7	-0.36	199.2	5.10
CAEYM	%	-	5.00	-	2.23	-	15.21

Table 4.13 Results from the MOO experiment using the RS with 1050 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2268	-0.43	6.3385	-2.23	6.4422	-3.91
$t_{BottomWall}$	mm	5.9392	-0.66	5.9603	-1.02	6.0992	-3.38
$t_{FrontWall}$	mm	5.7399	-0.70	5.7603	-1.06	5.8771	-3.11
$t_{BackWall}$	mm	6.1390	-0.64	6.1659	-1.08	6.2885	-3.09
$t_{RightWall}$	mm	5.8385	-0.66	5.8634	-1.09	5.9969	-3.39
$t_{LeftWall}$	mm	6.3697	-1.11	6.4369	-2.17	6.4291	-2.05
CAE	%	-	4.21	-	8.66	-	18.92
Young Modulus	GPa	205.9	1.90	203.7	2.95	190.1	9.41
CAEYM	%	-	6.11	-	11.61	-	28.33

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2453	-0.73	6.2168	-0.27	6.2309	-0.50
$t_{BottomWall}$	mm	5.9509	-0.86	5.9436	-0.74	5.9511	-0.87
$t_{FrontWall}$	mm	5.7791	-1.39	5.7391	-0.69	5.7791	-1.39
$t_{BackWall}$	mm	6.1416	-0.68	6.1356	-0.58	6.1376	-0.62
$t_{RightWall}$	mm	5.8486	-0.84	5.8504	-0.87	5.8490	-0.84
$t_{LeftWall}$	mm	6.3477	-0.76	6.3433	-0.69	6.3477	-0.76
CAE	%	-	5.26	-	3.84	-	4.97
Young Modulus	GPa	204.6	2.51	205.4	2.14	204.6	2.51
CAEYM	%	-	7.77	-	5.98	-	7.48

Table 4.14 Optimal objective function value for the SOO experiments using the RSs with 525 and 1050 design points.

Single Objective Optimization		
Number of Design Points	525	1050
Sum Sq. Diff. (mm^2)	6.19×10^{-5}	1.83×10^{-4}

Table 4.15 Sum of the squared differences for deformations in the MOO experiment using the RS with 525 design points.

Multiple Objective Optimization						
Candidate Solution		1	2		3	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.6998	3.99×10^{-5}	1.6707	5.18×10^{-4}	1.6886	2.40×10^{-5}
Bottom	2.1888	3.34×10^{-9}	2.1884	1.13×10^{-7}	2.1887	8.38×10^{-10}
Front	1.6989	1.97×10^{-6}	1.6912	4.02×10^{-5}	1.6886	7.95×10^{-5}
Back	1.8198	6.81×10^{-4}	1.9134	4.56×10^{-3}	1.8099	1.30×10^{-3}
Right	2.1661	8.99×10^{-7}	2.1630	1.61×10^{-5}	2.1672	4.84×10^{-8}
Left	1.7159	6.72×10^{-5}	1.7580	2.53×10^{-3}	1.6808	7.22×10^{-4}
Sum Sq. Diff. (mm²)	-	7.91×10^{-4}	-	7.66×10^{-3}	-	2.12×10^{-3}

Multiple Objective Optimization						
Candidate Solution		4	5		6	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.6997	3.86×10^{-5}	1.7074	1.94×10^{-4}	1.7380	1.98×10^{-3}
Bottom	2.1885	5.21×10^{-8}	2.1891	1.56×10^{-7}	2.1891	1.64×10^{-7}
Front	1.6891	7.09×10^{-5}	1.6883	8.55×10^{-5}	1.7009	1.16×10^{-5}
Back	1.8161	8.90×10^{-4}	1.8097	1.31×10^{-3}	1.8620	2.59×10^{-4}
Right	2.1611	3.52×10^{-5}	2.1671	6.92×10^{-9}	2.1673	6.34×10^{-8}
Left	1.7108	9.60×10^{-6}	1.7379	9.14×10^{-4}	1.6886	3.63×10^{-4}
Sum Sq. Diff. (mm²)	-	1.04×10^{-3}	-	2.51×10^{-3}	-	2.62×10^{-3}

Table 4.16 Sum of the squared differences for deformations in the MOO experiment using the RS with 1050 design points.

Multiple Objective Optimization						
Candidate Solution	1		2		3	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.7063	1.64×10^{-4}	1.6425	2.60×10^{-3}	1.6820	1.32×10^{-4}
Bottom	2.1886	7.67×10^{-9}	2.1888	1.06×10^{-8}	2.1891	1.88×10^{-7}
Front	1.6982	4.26×10^{-7}	1.7010	1.21×10^{-5}	1.7207	5.37×10^{-4}
Back	1.8474	2.39×10^{-6}	1.8441	3.41×10^{-6}	1.8629	2.91×10^{-4}
Right	2.1674	1.32×10^{-7}	2.1637	1.10×10^{-5}	2.1670	1.73×10^{-9}
Left	1.6854	4.98×10^{-4}	1.6520	3.10×10^{-3}	1.7753	4.57×10^{-3}
Sum Sq. Diff. (mm ²)	-	6.65×10^{-4}	-	5.73×10^{-3}	-	5.53×10^{-3}

Candidate Solution	4		5		6	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.7032	9.32×10^{-5}	1.7182	6.12×10^{-4}	1.7138	4.14×10^{-4}
Bottom	2.1891	1.56×10^{-7}	2.1893	3.87×10^{-7}	2.1889	4.54×10^{-8}
Front	1.6743	5.40×10^{-4}	1.7027	2.74×10^{-5}	1.6741	5.48×10^{-4}
Back	1.8567	1.18×10^{-4}	1.8550	8.33×10^{-5}	1.8604	2.09×10^{-4}
Right	2.1692	4.67×10^{-6}	2.1592	6.02×10^{-5}	2.1686	2.72×10^{-6}
Left	1.7134	3.20×10^{-5}	1.7106	8.50×10^{-6}	1.7133	3.12×10^{-5}
Sum Sq. Diff. (mm ²)	-	7.87×10^{-4}	-	7.91×10^{-4}	-	1.20×10^{-3}

consulted in appendices A and B. These metrics, calculated for all RSO experiments, are presented in figures 4.20, 4.21 and 4.22. Notice that the information concerning the RSs with 125 and 725 design points is presented in table 4.17, and the information regarding the optimization procedures implemented on both RSs is presented in table 4.18.

Table 4.17 Parameters for generating a RS with 125 design points and a RS with 725 design points and relative errors obtained after creating them.

Number of Design Points in the RS	125	725
Initial number of design points	16	16
Number of validation points	32	32
Maximum Relative Error (%)	0.1	0.1
Maximum Depth	10	10
Maximum Number of Refinement Points	125	725
Relative Error (%)	4.8683	0.3010

Table 4.18 Criteria used for the optimization procedure in the RSOs using 125 and 725 design points.

Criteria	Value
Estimated number of design points	33600
Initial samples	7000
Samples per iteration	1400
Maximum Pareto percentage	70
Convergence stability percentage	0
Maximum number of iterations	20

Based on the aforementioned graphs, one can state that the RSO using a RS with 125 design points presents suspiciously good results, particularly for SOO. It would be expected that the results obtained from RSO would be increasingly better as the number of design points used to create the RS increases, however this trend does not seem to be followed. That being said, the four RSs were compared to assess their quality and investigate the credibility of the obtained results.

Table 4.19 presents the candidate solution obtained from the SOO using the RS with 125 design points. To assess the credibility of this solution, the candidate solution's wall thicknesses and Young Modulus were used to obtain the corresponding wall deformations according to each of the four RSs (using 125, 525, 725 and 1050 design points) and according to the FE model. Additionally, the error between each wall's deformation in the RS and the FE model (equation 4.8) was calculated, as well as the CAE (equation 4.9), which is the sum of the absolute value of each wall's errors. The results obtained are presented in table 4.20.

$$Error_{FE} (\%) = \frac{\delta_{RS} - \delta_{FEModel}}{\delta_{FEModel}} \quad (4.8)$$

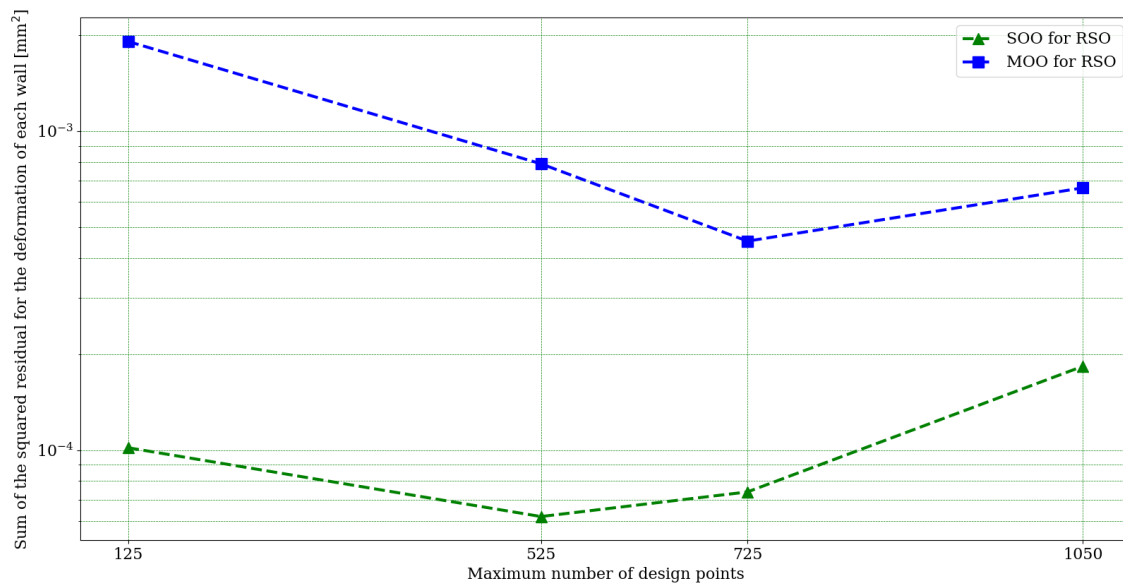


Figure 4.20 General comparison of the sum of the squared differences between the deformations in the updated model and those in the synthetic data for RSO.

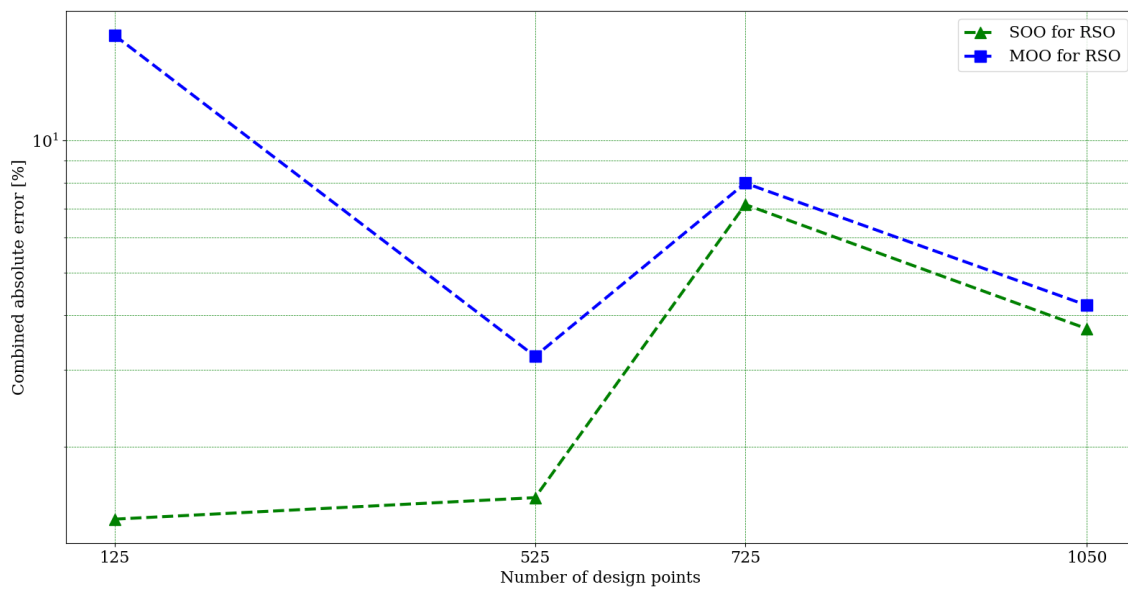


Figure 4.21 General comparison of the CAE calculated for all conducted experiments using RSO.

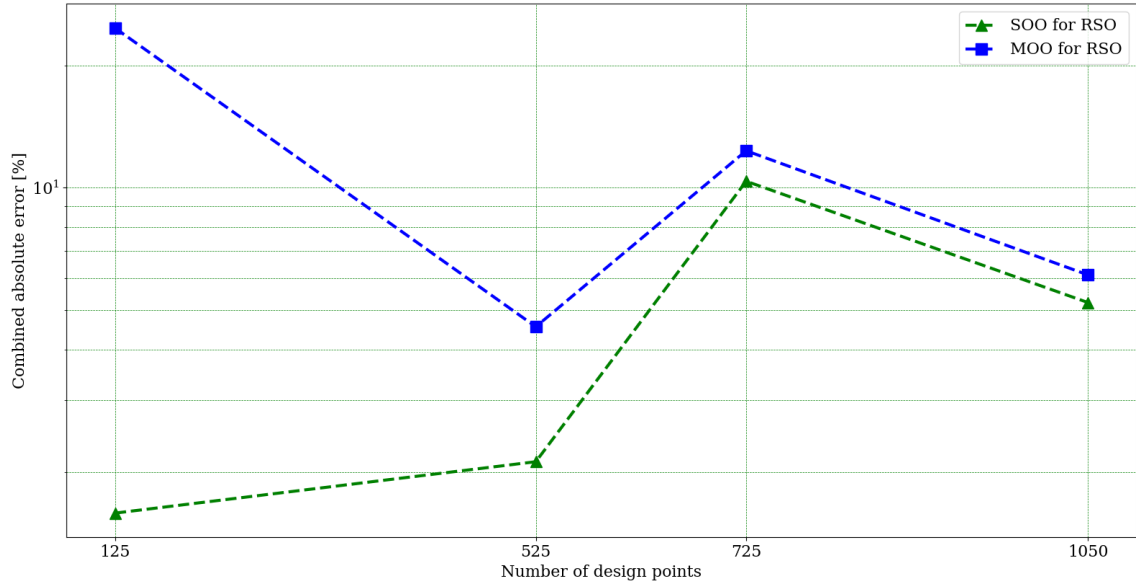


Figure 4.22 General comparison of the CAEYM calculated for all conducted experiments using RSO.

Table 4.19 Candidate solution obtained from the SOO using the RS with 125 design points and corresponding deformation evaluation in the FE model.

	Unit	t_{RS125}	$\delta_{FE Model}$
Top Wall	mm	6.2155	1.6863
Bottom Wall	mm	5.9145	2.1786
Front Wall	mm	5.7082	1.6954
Back Wall	mm	6.1157	1.8368
Right Wall	mm	5.8139	2.1574
Left Wall	mm	6.3146	1.7005
Young Modulus	GPa	209.4	-

$$\begin{aligned} \text{Combined Absolute Error}_{FE}(\%) &= CAEFE(\%) \\ &= \sum_{n=1}^6 \left| \frac{\delta_{RS,n} - \delta_{FEModel,n}}{\delta_{FEModel,n}} \right| \end{aligned} \quad (4.9)$$

Table 4.20 Wall deformations for the candidate solution obtained from the SOO using a RS with 125 design points according to each of the four RSs (using 125, 525, 725 and 1050 design points) and according to the FE model and corresponding errors.

RS	125 Design Points		525 Design Points	
	δ (mm)	$Error_{FE}(\%)$	δ (mm)	$Error_{FE}(\%)$
Top Wall	1.6897	0.20	1.6876	0.07
Bottom Wall	2.1855	0.31	2.1806	0.09
Front Wall	1.6954	0.00	1.6957	0.02
Back Wall	1.8425	0.31	1.8379	0.06
Right Wall	2.1649	0.35	2.1590	0.07
Left Wall	1.7001	-0.02	1.7011	0.04
CAEFE (%)	-	1.20	-	0.35

RS	725 Design Points		1050 Design Points	
	δ (mm)	$Error_{FE}(\%)$	δ (mm)	$Error_{FE}(\%)$
Top Wall	1.6866	0.02	1.6864	0.01
Bottom Wall	2.1790	0.02	2.1789	0.01
Front Wall	1.6957	0.02	1.6957	0.02
Back Wall	1.8373	0.03	1.8372	0.02
Right Wall	2.1581	0.03	2.1580	0.03
Left Wall	1.7008	0.02	1.7007	0.01
CAEFE (%)	-	0.13	-	0.10

As can be seen from table 4.20, the results obtained from the RSO using the RS with 125 design points present a greater error when compared to the deformations obtained from the FE model. Bearing in mind that the RS provides an estimate for the deformations the FE model would calculate for a given set of wall thicknesses and Young Modulus, this means that the RS with 125 design points provides a meta-model of the relation between walls thicknesses, deformations and Young Modulus which, despite having an higher approximation error, may yield candidate points with predicted sum of the squared residuals lower than it's actual value (when recalculated with the FEM model). That being said, the results presented in table 4.20 reveal that the RS with 1050 design points creates a better meta-model (that is, a better approximation of the FE model) than the RSs with less design points.

Besides the appraisal presented above, it is also relevant to note that the error for

each of the walls deformations does not pass the maximum error presented by *Anslys* for any of the RSs ($Maximum\ Error_{RS125} = 4.868\%$, $Maximum\ Error_{RS525} = 0.498\%$, $Maximum\ Error_{RS725} = 0.301\%$, $Maximum\ Error_{RS1050} = 0.153\%$).

So, as can be seen from this assessment, it is necessary to take into account the error introduced by the quality of the RSs into the obtained solutions. Particularly, it is relevant to note that the imprecision in the RSs may lead to apparently better results, which do not reflect the actual behaviour of the FE model and, consequently, the real structure. Therefore, to create a fair comparison between all the approaches presented in this study, each candidate solution was reevaluated in the FE model (using the thicknesses obtained from the optimization procedures for each candidate solution) and the deformations obtained from that reevaluation were used to calculate a new sum of the squared differences. This metric measures the squared residual between the wall deformations for the candidate solution for each procedure evaluated in the FE model and the target deformations established by the synthetic data. Figure 4.23 presents the reevaluated sum of the squared differences for all the RSO approaches used in this study.

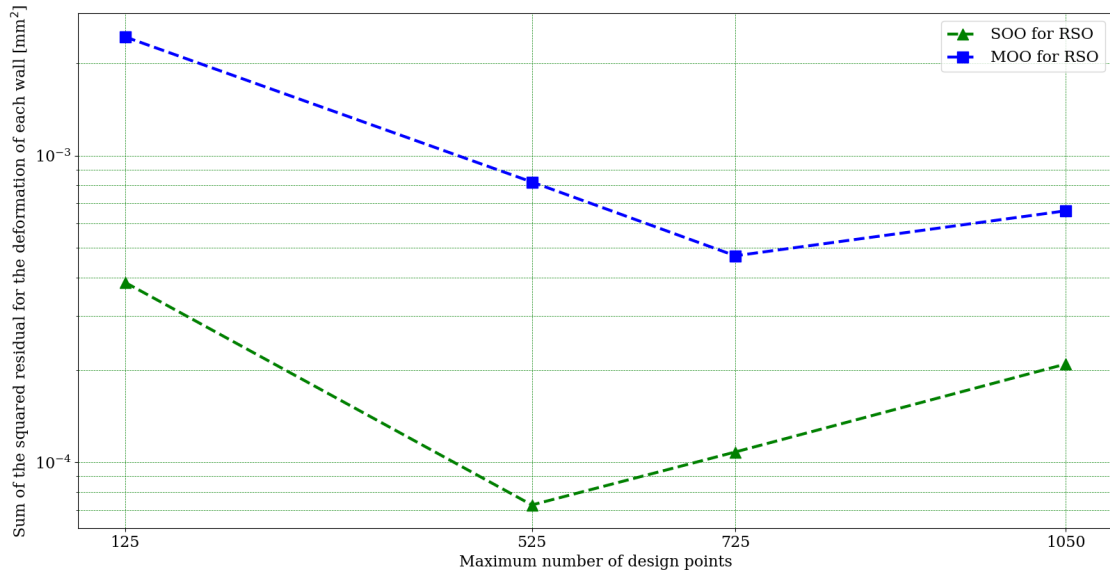


Figure 4.23 General comparison of the sum of the squared differences between the deformations in the updated model, calculated with the FE model, and those in the synthetic data for RSO.

After analysing figure 4.23, it seems that an increase in the number of design points used to generate the RS may lead to improved results. However, this trend is neither obvious nor followed by all the evaluated points. Anyhow, one can say that the increase in the number of design points leads to a decrease in the error between the RS and the synthetic FE model. Initially, this occurs in a global or generalized manner, which causes a sharp decay of the sum of the residuals. Once the surface is reasonably approximated globally, however, the accuracy around the design point which represents the cube from

the synthetic model becomes more significant. Therefore, further refining the surface in areas away from this design point can degrade the accuracy of the response surface around this design point, which is the global optimum. Notice, however, that this depends largely on the interpolation technique used in the response surface (as is shown in chapter 5). Nonetheless, the increase in the sum of the squared differences displays a less pronounced growth after 525/725 design points, validating the potential theory that this is a consequence of loss of local precision due to the inclusion of more design points in remote areas.

Furthermore, one should also bear in mind the fact that the optimization algorithm used for both optimization approaches, NSGA-II, is a non-deterministic algorithm which offers no guarantee of reaching the optimal value in all of its runs and may mistake local optima for global optima, leading to unsatisfactory results. MOO is particularly prone to displaying this sort of behaviour since it deals with multiple objectives, more often than not conflicting, and approaching one objective may negatively influence the attempt to approach another objective. Therefore, the results obtained from MOO are more susceptible to unpredictability and error. This unfortunate effect is displayed in figure 4.24, where one can see that not all objectives are approached at the same rate nor reach the same error level. It is interesting to note that, even though SOO and MOO run an equal number of evaluations and iterations, SOO leads to stability percentages around 5 times lower, which means that the latter optimization method presents a stabler set of candidate solutions. However, the Pareto percentage seems to be similar for all of the experiences.

All in all, it would appear that MOO converges more abruptly, thus the higher stability percentage, but leads to worse results. Furthermore, it is a less computationally effective technique, since it requires satisfying diverse objectives simultaneously and ranking candidate solutions according to a dominance criteria. Therefore, from this simple analysis, it would appear that SOO is superior to MOO, though further testing may be necessary to extrapolate this conclusion to more complex problems.

4.3.3 Direct optimization (DO)

Contrary to RSO, DO runs NSGA-II directly on the model. This means that, for each evaluation the algorithm performs, the FE model must be re-evaluated, leading to a slower optimization process. Therefore, the main difference between DO and RSO is the method used for the evaluation performed by the algorithm, which for the latter simply means consulting the RS. The optimization procedure is then fairly similar for both methods. The criteria for the DO is simply defined in the DO block, which is represented in figure 4.25.

Note that, since the GA is not deterministic, different runs of the optimization process may lead to different results, specially when using a small population. This notion will be taken into account later on.

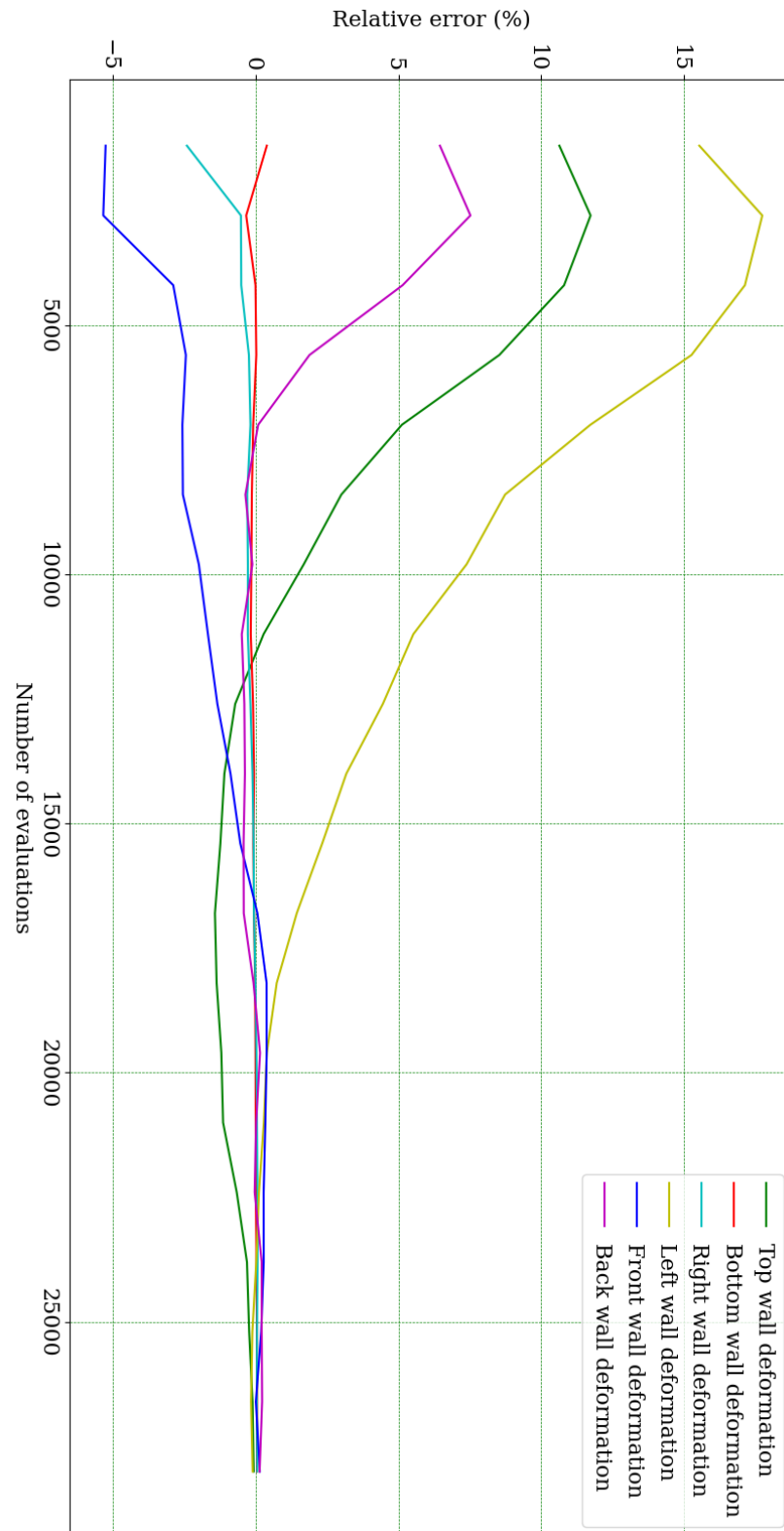


Figure 4.24 Evolution of the relative error of the average deformation value per iteration for MOO on a RS created using 1050 design points.

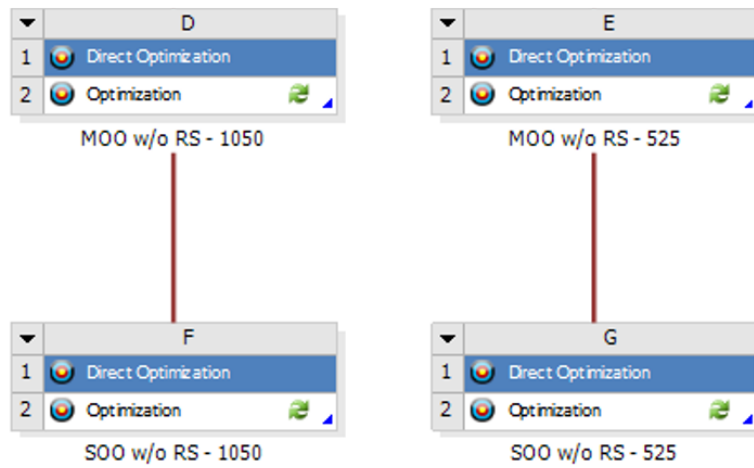


Figure 4.25 DO blocks in the *Ansys Workbench*.

Since the optimization algorithm used for DO is the same one used for the RSO, NSGA-II, the criteria remains the same:

- Estimated number of design points
- Number of initial samples
- Number of samples per iteration
- Maximum allowable Pareto percentage
- Convergence stability percentage
- Maximum number of iterations
- Maximum number of candidates

4.3.3.1 Running the optimization procedures

In order to effectively compare the solutions obtained from DO with those obtained from RSO, two experiments were conducted. The criteria used for each of them is presented in table 4.21. Notice that the estimated number of design points used for each of the DO experiments is equal do the maximum number of refinement points used for creating the RSs in the RSO experiments. This means that, for both cases, the FE model must be conducted either 525 or 1050 times, creating a fair ground for performance comparison later on. Notice that here, similarly to what was done for RSO, the convergence stability percentage was set o 0, forcing the simulations to run all iterations (assuming that the maximum Pareto percentage is not reached).

For each of the two aforementioned experiments, the SOO and MOO approaches were both used. These approaches were defined in a similar manner to that used for the RSO, as can be seen in figures 4.18 and 4.19, presented in subsection 4.3.2.

Table 4.21 Criteria for DO experiments.

Experiment	525 design points	1050 design points
Estimated number of design points	525	1050
Initial samples	50	100
Samples per iteration	25	50
Maximum Pareto percentage	70	70
Convergence stability percentage	0	0
Maximum number of iterations	20	20

4.3.3.2 Results and comments

The results obtained from the DO experiments are presented in tables 4.22 to 4.28 and figures 4.26, 4.27 and 4.28. Figures 4.26 and 4.27 present the relative error for the average deformation of each wall per iteration as a function of the number of evaluated points, while figure 4.28 compares the evolution of the relative error in terms of the number of evaluations (design points) for SOO based on the approaches using 525 and 1050 design points. Notice that, when 525 design points are used, the graphs start from 25 evaluations, as that is the number of individuals from the initial population that get selected to reproduce and integrate the next generation. For the same reasons, when 1050 design points are used, the graphs start from an initial set of 50 individuals, with the number of evaluations increasing at a rate of 50 evaluations per iteration. Similarly to the information provided in subsection 4.3.2, tables 4.26 to 4.28 refer to the analysis of the sum of the squared differences between the updated deformations and the synthetic data deformations.

Table 4.22 Details of the SOO and MOO experiments using DO.

Type of Optimization	SOO		MOO	
Number of Design Points	525	1050	525	1050
No. of iterations:	20	20	20	20
Sample size:	25	50	25	50
No. of evaluations:	514	1028	514	1028
Stability percentage:	0.0333	0.0221	0.6547	3.4412
Pareto percentage:	4	2	4	2

Unlike RSO, and according to the CAE, CAEYM and the sum of the squared differences, DO only seems to perform better when SOO is used for a smaller number of design points, as can be seen in table 4.26. MOO, on the other hand, seems to follow the expected trend across all metrics, that is, a lower value is achieved with a greater number of design points, as can be seen in tables 4.24, 4.25, 4.27, 4.28. Therefore, an increase in the number of design points seems to benefit the results for MOO but harm them for SOO, which is an unexpected result. For the case of 525 design points, the results obtained from SOO are clearly better than those obtained from MOO, presenting a CAE around 4 times lower,

Table 4.23 Results from the SOO experiments using DO with 525 and 1050 design points.

Single Objective Optimization					
Number of Design Points		525		1050	
Parameter	Unit	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.1882	0.19	6.5186	-5.14
$t_{BottomWall}$	mm	5.8969	0.05	6.1815	-4.77
$t_{FrontWall}$	mm	5.6916	0.15	5.9157	-3.78
$t_{BackWall}$	mm	6.0409	0.97	6.3627	-4.31
$t_{RightWall}$	mm	5.7690	0.53	6.0762	-4.76
$t_{LeftWall}$	mm	6.2802	0.31	6.5368	-3.76
CAE	%	-	2.21	-	26.52
Young Modulus	GPa	211.6	-0.83	184.7	12.01
CAEYM	%	-	3.04	-	38.53

Table 4.24 Results from the MOO experiment using DO with 525 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2267	-0.43	6.2268	-0.43	6.2268	-0.43
$t_{BottomWall}$	mm	6.0472	-2.49	6.0870	-3.17	6.0865	-3.16
$t_{FrontWall}$	mm	5.8018	-1.79	5.8029	-1.80	5.8021	-1.79
$t_{BackWall}$	mm	5.7687	5.43	5.8061	4.82	5.8061	4.82
$t_{RightWall}$	mm	5.7882	0.20	5.7889	0.19	5.7883	0.20
$t_{LeftWall}$	mm	6.2616	0.61	6.2652	0.55	6.2614	0.61
CAE	%	-	10.96	-	10.97	-	11.02
Young Modulus	GPa	211.3	-0.68	211.3	-0.69	211.3	-0.68
CAEYM	%	-	11.64	-	11.65	-	11.70

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2274	-0.44	6.2268	-0.43	6.2268	-0.43
$t_{BottomWall}$	mm	6.0861	-3.15	6.0870	-3.17	6.0861	-3.15
$t_{FrontWall}$	mm	5.8021	-1.79	5.8082	-1.90	5.8091	-1.91
$t_{BackWall}$	mm	5.8069	4.80	5.8123	4.72	5.8057	4.82
$t_{RightWall}$	mm	5.7882	0.20	5.7889	0.19	5.7889	0.19
$t_{LeftWall}$	mm	6.2616	0.61	6.2652	0.55	6.2649	0.56
CAE	%	-	11.00	-	10.96	-	11.07
Young Modulus	GPa	211.3	-0.69	211.3	-0.69	211.3	-0.68
CAEYM	%	-	11.69	-	11.64	-	11.76

Table 4.25 Results from the MOO experiment using DO with 1050 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2445	-0.72	6.2013	-0.02	6.2008	-0.01
$t_{BottomWall}$	mm	5.9832	-1.41	5.9953	-1.61	5.9810	-1.37
$t_{FrontWall}$	mm	5.8185	-2.08	5.8206	-2.12	5.8184	-2.08
$t_{BackWall}$	mm	6.1257	-0.42	6.1111	-0.18	6.1172	-0.28
$t_{RightWall}$	mm	5.8939	-1.62	5.8944	-1.63	5.8938	-1.62
$t_{LeftWall}$	mm	6.4456	-2.31	6.4463	-2.32	6.4519	-2.41
CAE	%	-	8.56	-	7.88	-	7.77
Young Modulus	GPa	199.9	4.77	200.3	4.59	200.4	4.52
CAEYM	%	-	13.33	-	12.47	-	12.29

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2008	-0.01	6.2013	-0.02	6.2013	-0.02
$t_{BottomWall}$	mm	5.9810	-1.37	5.9810	-1.37	5.9810	-1.37
$t_{FrontWall}$	mm	5.8184	-2.08	5.8204	-2.11	5.8205	-2.11
$t_{BackWall}$	mm	6.1172	-0.28	6.1005	-0.01	6.0961	0.06
$t_{RightWall}$	mm	5.8938	-1.62	5.8938	-1.62	5.8938	-1.62
$t_{LeftWall}$	mm	6.4526	-2.42	6.4504	-2.39	6.4463	-2.32
CAE	%	-	7.78	-	7.52	-	7.51
Young Modulus	GPa	200.4	4.52	200.3	4.59	200.3	4.59
CAEYM	%	-	12.30	-	12.11	-	12.10

Table 4.26 Optimal objective function value for the SOO experiment using DO with 525 design points.

Single Objective Optimization		
Number of Design Points	525	1050
Sum Sq. Diff. (mm^2)	2.13×10^{-3}	4.25×10^{-3}

Table 4.27 Sum of the squared differences for deformations in the MOO experiment using DO with 525 design points.

Multiple Objective Optimization						
Candidate Solution	1		2		3	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.6638	8.81×10^{-4}	1.6638	8.81×10^{-4}	1.6639	8.74×10^{-4}
Bottom	2.0231	2.74×10^{-2}	1.9847	4.16×10^{-2}	1.9853	4.14×10^{-2}
Front	1.6015	9.22×10^{-3}	1.6011	9.30×10^{-3}	1.6018	9.17×10^{-3}
Back	2.1686	1.04×10^{-1}	2.1268	7.89×10^{-2}	2.1270	7.90×10^{-2}
Right	2.1648	4.75×10^{-6}	2.1639	9.90×10^{-6}	2.1647	5.20×10^{-6}
Left	1.7263	3.47×10^{-4}	1.7233	2.42×10^{-4}	1.7265	3.53×10^{-4}
Sum Sq. Diff. (mm ²)	-	1.42×10^{-1}	-	1.31×10^{-1}	-	1.31×10^{-1}
Candidate Solution	4		5		6	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.6634	9.08×10^{-4}	1.6637	8.86×10^{-4}	1.6638	8.83×10^{-4}
Bottom	1.9855	4.13×10^{-2}	1.9846	4.17×10^{-2}	1.9854	4.13×10^{-2}
Front	1.6017	9.18×10^{-3}	1.5967	1.02×10^{-2}	1.5960	1.03×10^{-2}
Back	2.1260	7.84×10^{-2}	2.1200	7.51×10^{-2}	2.1273	7.92×10^{-2}
Right	2.1647	5.41×10^{-6}	2.1638	1.02×10^{-5}	2.1639	9.66×10^{-6}
Left	1.7262	3.42×10^{-4}	1.7232	2.41×10^{-4}	1.7235	2.49×10^{-4}
Sum Sq. Diff. (mm ²)	-	1.30×10^{-1}	-	1.28×10^{-1}	-	1.32×10^{-1}

Table 4.28 Sum of the squared differences for deformations in the MOO experiment using DO with 1050 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)
Top	1.7434	2.49×10^{-3}	1.7736	6.41×10^{-3}	1.7726	6.26×10^{-3}	
Bottom	2.2040	2.34×10^{-4}	2.1868	3.77×10^{-6}	2.2004	1.38×10^{-4}	
Front	1.6813	2.62×10^{-4}	1.6758	4.70×10^{-4}	1.6764	4.46×10^{-4}	
Back	1.9156	4.86×10^{-3}	1.9253	6.30×10^{-3}	1.9181	5.22×10^{-3}	
Right	2.1698	7.98×10^{-6}	2.1647	5.51×10^{-6}	2.1639	9.63×10^{-6}	
Left	1.6757	1.02×10^{-3}	1.6717	1.30×10^{-3}	1.6662	1.72×10^{-3}	
Sum Sq. Diff. (mm²)	-	8.88×10^{-3}	-	1.45×10^{-2}	-	1.38×10^{-2}	

Multiple Objective Optimization							
Candidate Solution		4		5		6	
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)
Top	1.7726	6.26×10^{-3}	1.7736	6.41×10^{-3}	1.7735	6.41×10^{-3}	
Bottom	2.2004	1.38×10^{-4}	2.2021	1.78×10^{-4}	2.2020	1.78×10^{-4}	
Front	1.6764	4.45×10^{-4}	1.6759	4.68×10^{-4}	1.6757	4.73×10^{-4}	
Back	1.9181	5.22×10^{-3}	1.9354	8.01×10^{-3}	1.9395	8.77×10^{-3}	
Right	2.1639	9.63×10^{-6}	2.1655	2.25×10^{-6}	2.1654	2.63×10^{-6}	
Left	1.6657	1.76×10^{-3}	1.6686	1.53×10^{-3}	1.6717	1.29×10^{-3}	
Sum Sq. Diff. (mm²)	-	1.38×10^{-2}	-	1.66×10^{-2}	-	1.71×10^{-2}	

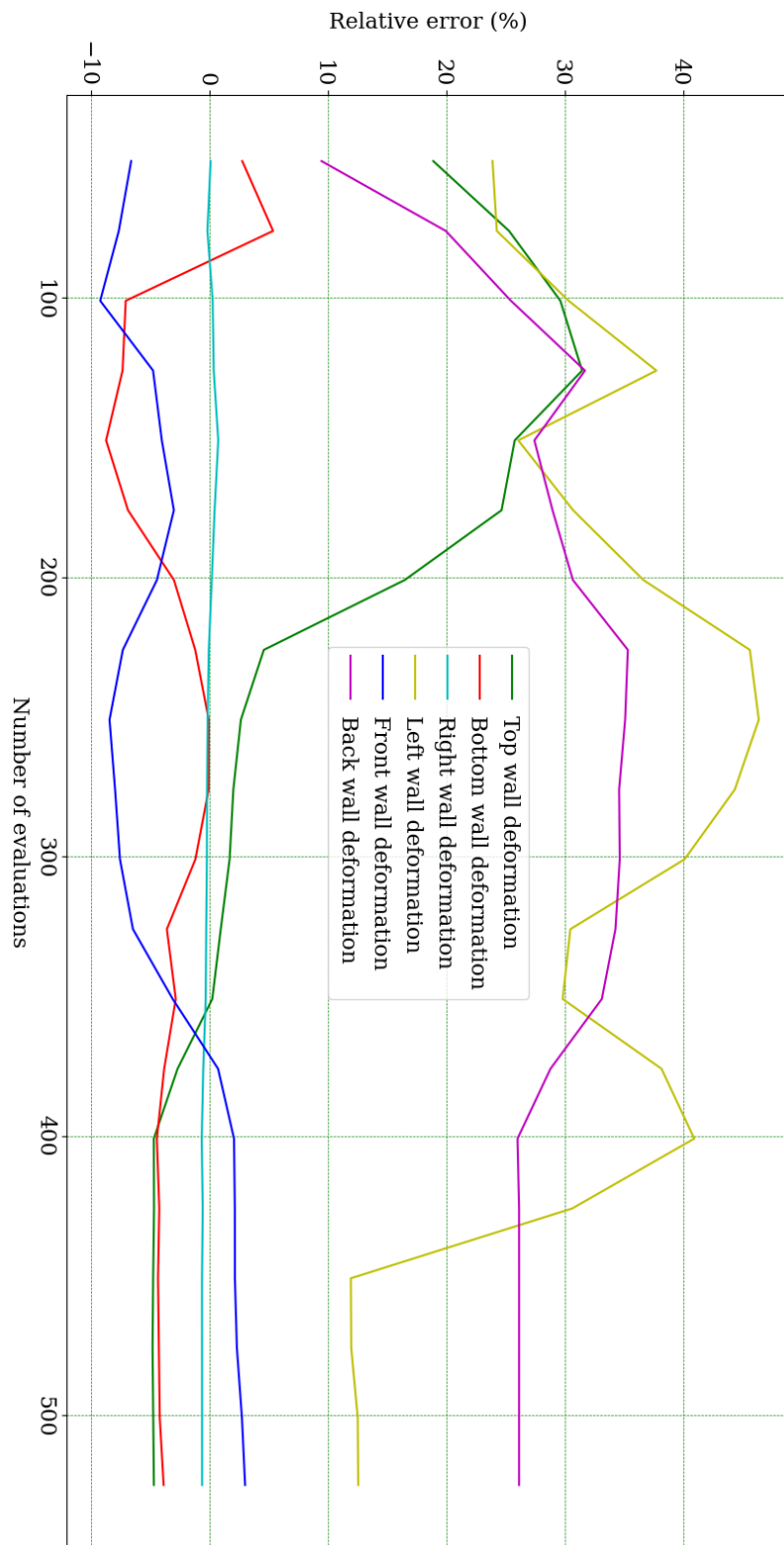


Figure 4.26 Evolution of the relative error of the average deformation value per iteration for DO using MOO with 525 design points.

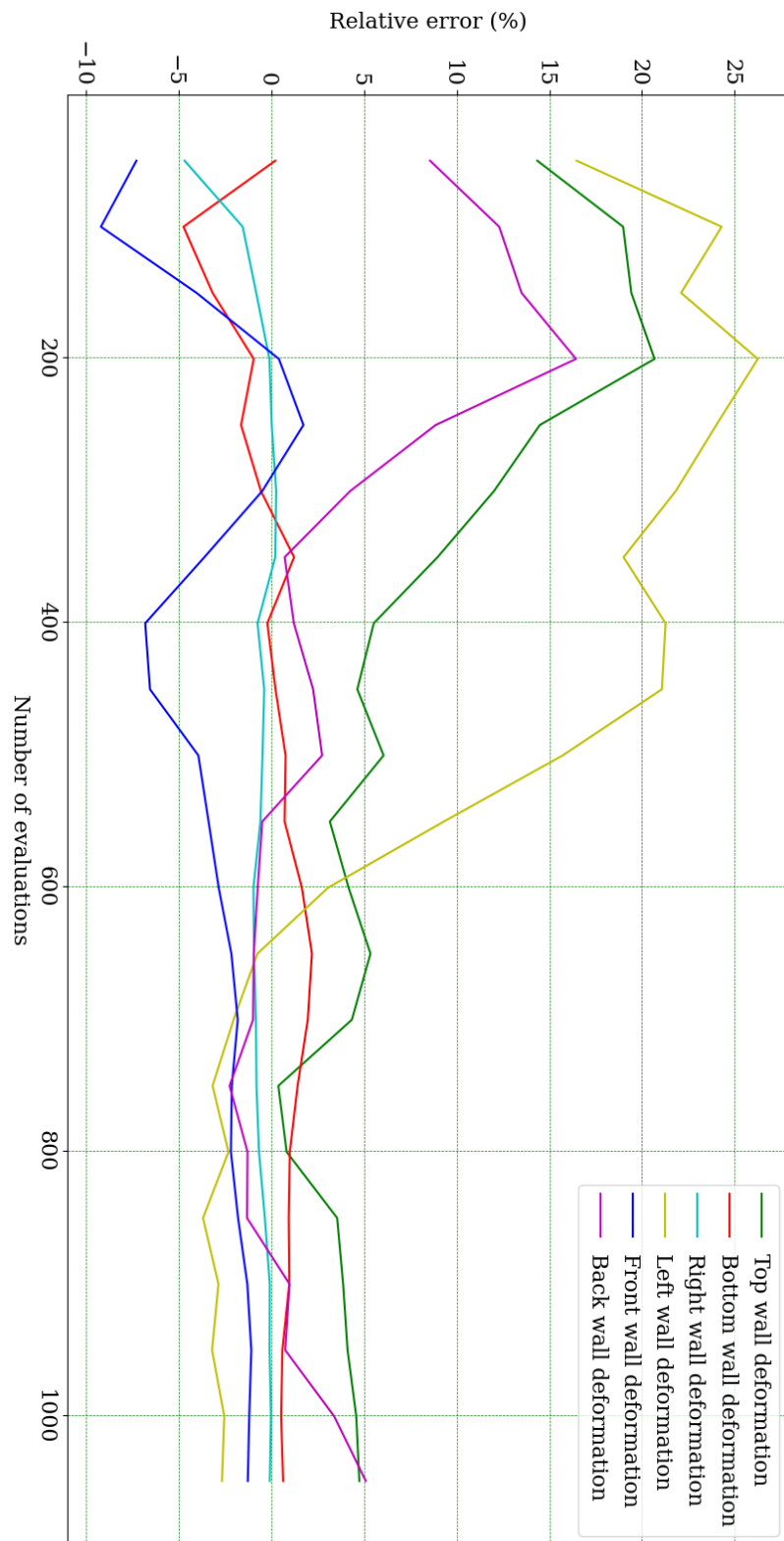


Figure 4.27 Evolution of the relative error of the average deformation value per iteration for DO using MOO with 1050 design points.

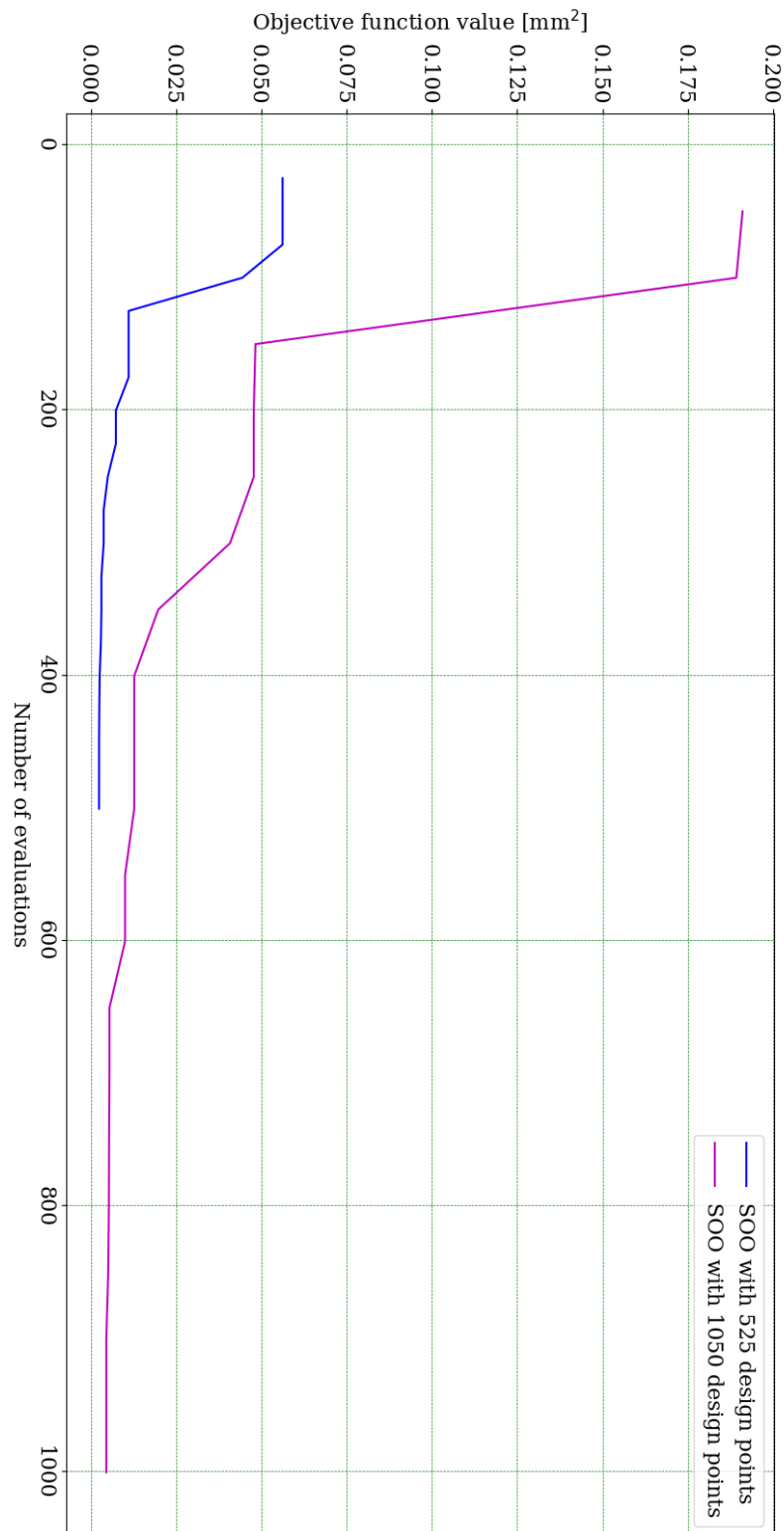


Figure 4.28 Comparison of the evolution of the objective function value as a function of the number of evaluations for DO using SOO.

both when considering and not considering the Young Modulus, and a sum of the squared difference around 100 times lower. However, when using 1050 design points, the results are surprising as both the CAE and CAEYM for the MOO are around 3 times lower than the ones calculated for the SOO. Nonetheless, the sum of the squared difference is still around 2 times lower for the SOO. The unexpected results found in the case of SOO using 1050 design points emphasize the stochastic nature of the optimization algorithm, NSGA-II, which does not guarantee that the optimal point is reached and may wrongly converge towards a local optimal point instead of the global optima. It is particularly interesting to note the fact that, for that experiment, the metrics of CAE and CAEYM, which quantify the quality of the approximation of the wall thicknesses and Young Modulus, present extremely poor results, however the sum of the squared differences, which quantifies the quality of the approximation of the wall deformations, does not present equally poor results. That being said, since the target was set for the wall deformations, a fairly satisfactory approximation for these values may not translate into a satisfactory approximation of the wall thickness and Young Modulus, thus emphasizing the need to evaluate the obtained results according to various metrics.

In any case, however, the use of a greater number of design points seems to benefit the performance of MOO, as can be seen in figure 4.28. This figure also presents an interesting insight into the evolution of the optimization algorithm, highlighting the fact that SOO using 525 design points starts the optimization process from a better candidate solution than the SOO using 1050 design points, converging earlier and leading to better results. SOO using 1050 design points seems to converge abruptly in the beginning, however it never reaches a candidate solution with an objective function value nearly as good as the one found for SOO with 525 design points. This representation further emphasizes the stochastic nature of the optimization algorithm, NSGA-II, and how that aspect may affect the obtained results in an unpredictable way. Contrary to what was found when generating the RSs, running an optimization procedure using a GA can lead to different (and even unexpected) results for each run of the same optimization experiment. Therefore, it is not sensible to extrapolate the obtained results and conclusions to different and more complex problems without a wider range of experiments which can shed light into the intricacies of the algorithm's behaviour.

Interestingly, SOO leads to a higher stability percentage for the case of DO using 525 design points, but a lower one for the 1050 design points. This means that the set of candidate solutions found for SOO using 525 design points is more heterogeneous. Additionally, the Pareto percentage is higher for 525 design points than for 1050 for both SOO and MOO, which means that a greater percentage of the sample set is part of the Pareto front.

Taking into account the information presented in the graphs from figures 4.26 and 4.27, it seems that DO using MOO with 525 design points does not accurately approach the correct value for the deformation of each wall, presenting no relevant evolution after

450 design points. The left and back wall deformations seem to present the worst approximations, with a particularly irregular evolution of the relative error, which does not drop from 10% for the left wall and 20% for the back wall. The right wall deformation, however, seems to be accurately approached from the very beginning, suffering no relevant changes as the optimization evolves, while the remaining walls seem to have a steady drop in the relative error. This may mean that the top, bottom, front and right walls may have been prioritized in the optimization process. Bearing that in mind, the unsteady and irregular evolution of the relative error for the left and back walls may be a consequence of crossover between candidate solutions which present better approximations for the deformation of the remaining walls, but do not necessarily fit the targets for these two walls, generating candidate solutions which lead to decreased relative error for some walls and an erratic evolution of the relative error for the back and left walls. The MOO using 1050 design points, on the other hand, seems to present a better approach to the target value for each wall's deformation, leading to a lower relative error, with an absolute value no greater than 5%. In fact, when comparing the two graphs, MOO using 1050 design points seems to have generated a population with overall better candidate solutions, never presenting a relative error superior to 30% for any of the walls. This may be due to the fact that MOO using 1050 design points presents a larger population, which may positively affect diversity, which in turn allows for a greater coverage of the search space and a faster approach to optimal values. This difference in diversity is shown in figures 4.29 and 4.30 which display the evolution of the the candidate solutions from the optimization of the deformation of the back wall for DO using MOO using with 525 and 1050 design points, respectively. The greater diversity present in the MOO with 1050 design points is shown by the scattered points in the graph from figure 4.30, which represent the various individuals analysed throughout the optimization. The MOO with 525 design points, on the other hand, presents a less scattered graph, as can be seen in figure 4.29, indicating the aforementioned lack of diversity and early approach to a poor candidate solution. It is important to note that, since the GA is not deterministic, these graphs could be quite different if the simulation that generated them were to be conducted again. Therefore, it only seems fair to conclude that a greater number of samples per iteration benefits the diversity of the population and an increased number of evaluations allows the algorithm to approach the correct values, even if at a slower pace.

Taking this information into account, it is hard to conclude which approach is better, though it seems that, as the number of design points increases, MOO becomes a better option. For these reasons, further testing is still necessary to reach sustained conclusions and then extrapolate them to more complex problems.

4.3.4 Comparison between DO and RSO

In order to fairly compare DO and RSO one must analyse the number of design points used to create each RS and the number of evaluations conducted in the DO process, making

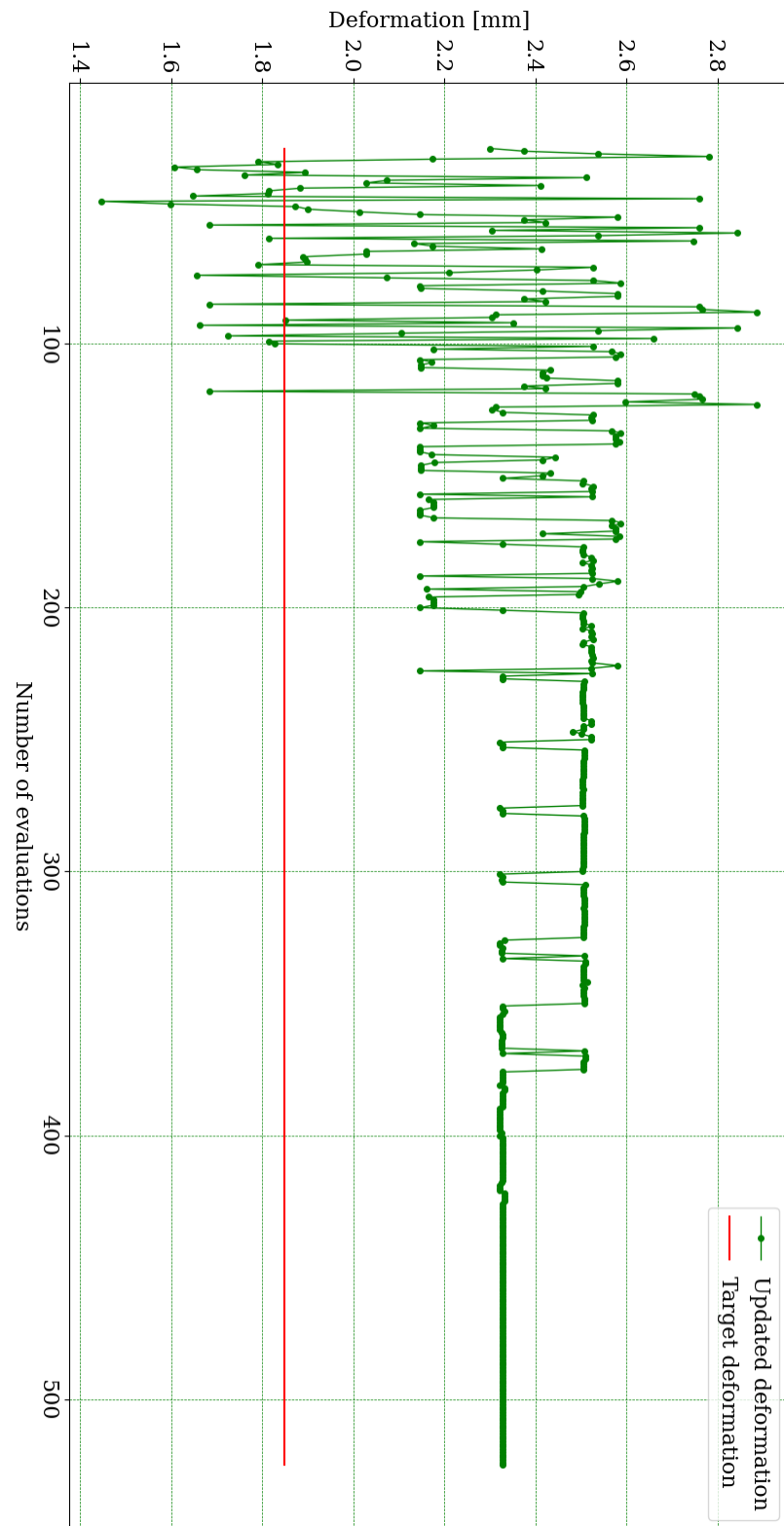


Figure 4.29 Evolution of the candidate solutions for the updated deformation of the back wall as a function of the number of evaluations for DO using MOO with 525 design points.

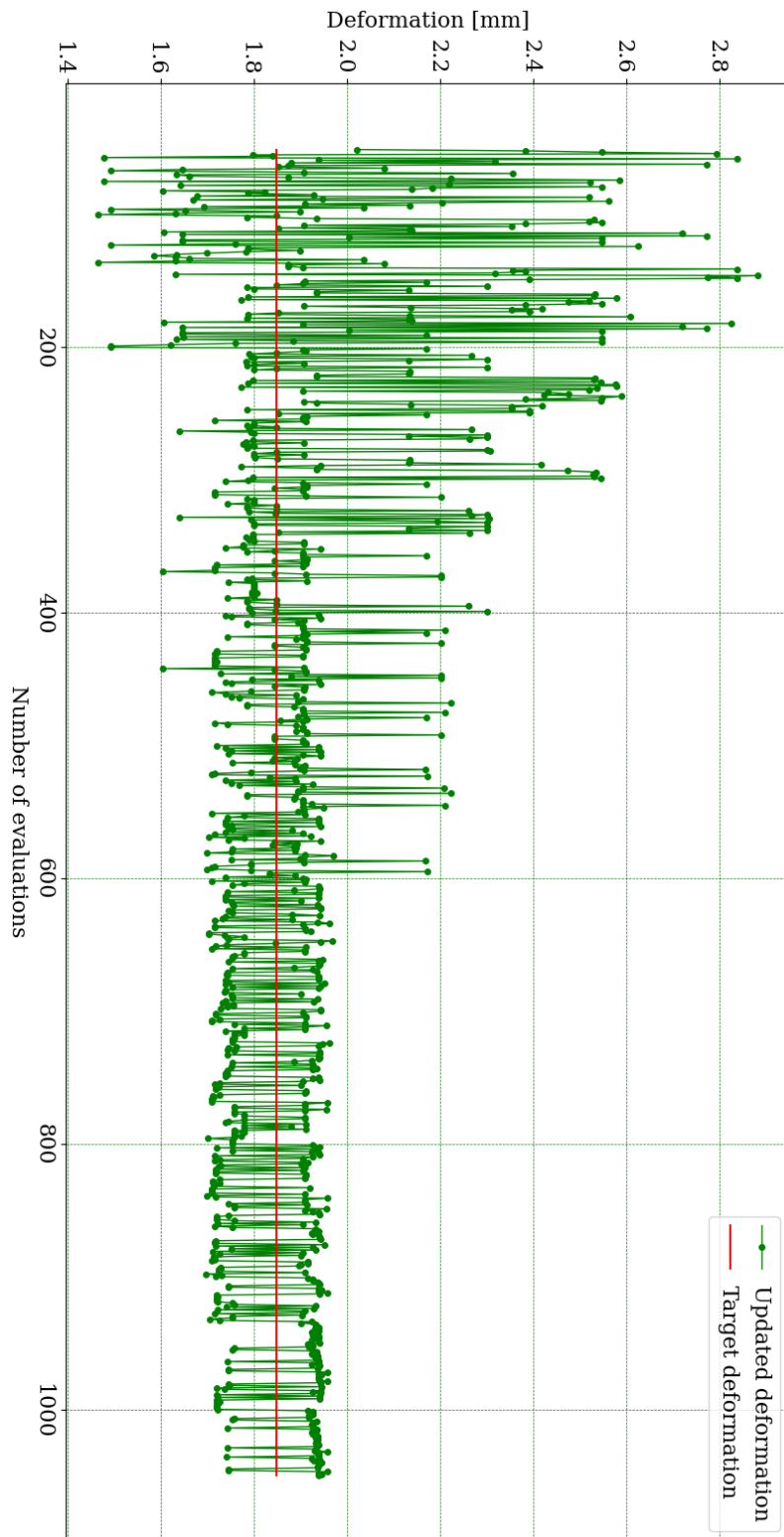


Figure 4.30 Evolution of the candidate solutions for the updated deformation of the back wall as a function of the number of evaluations for DO using MOO with 1050 design points.

sure that they are similar, thus representing an identical number of FE model evaluations, and may therefore be compared. Having set the convergence stability percentage to 0 for all the optimization procedures, it was possible to guarantee that the optimizations used the pre-determined number of design points, guaranteeing the aforementioned guideline. Additionally, the method for creating the RS is deterministic and used the complete set of design points determined initially. Therefore, the RS is not a source of deviation from the aforementioned guideline either.

To facilitate the comparison between the performance of all the aforementioned optimization procedures, three graphs were created which present the performance of all the methods used according to the three main metrics used in this study: sum of the squared residual for the deformation of each wall, CAE (combined absolute error concerning the walls' thickness) and CAEYM (combined absolute error concerning the walls' thickness and the Young Modulus). These graphs are presented in figures 4.31, 4.32 and 4.33. An additional graph is presented in figure 4.34, which presents the performance of all experiments in terms of the sum of the squared residual for the deformation of each wall, except here each candidate solution put forward by any RSO procedure has been reevaluated in the FE model, therefore avoiding an unfair comparison prompted by any candidate points with predicted sum of the squared residuals lower than its actual value (when recalculated with the FE model). This representation is in line with the analysis conducted in subsection 4.3.2.3.

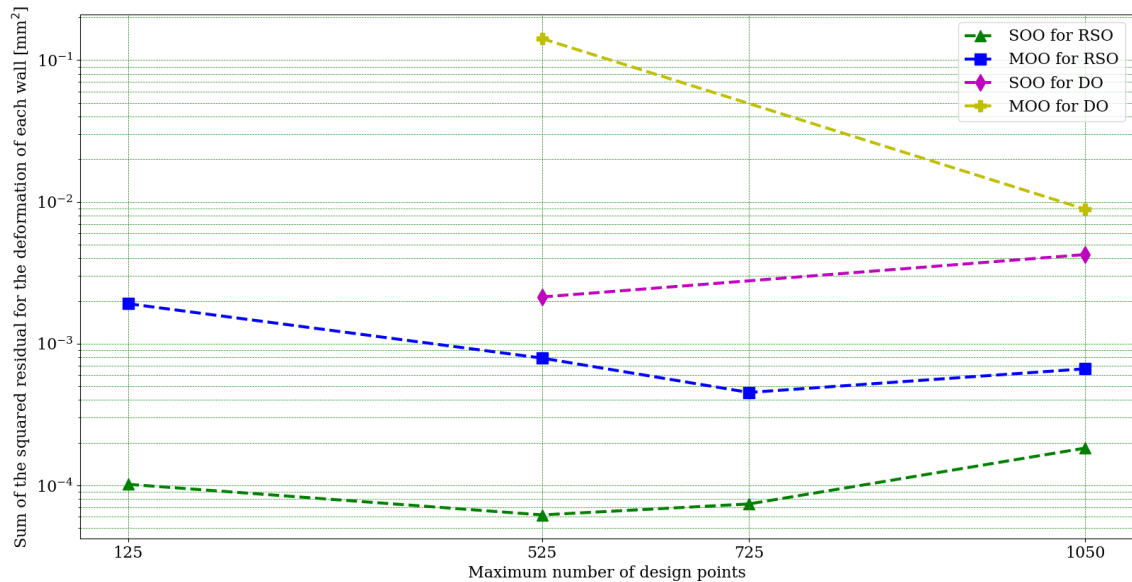


Figure 4.31 General comparison of the sum of the squared differences between the deformations in the updated model and those in the synthetic data.

Therefore, and based on the aforementioned graphs, one can state that, for the case of RSO using a RS with 1050 design points versus the case of DO using the same number of design points, both SOO and MOO optimization procedures for the RSO outperform

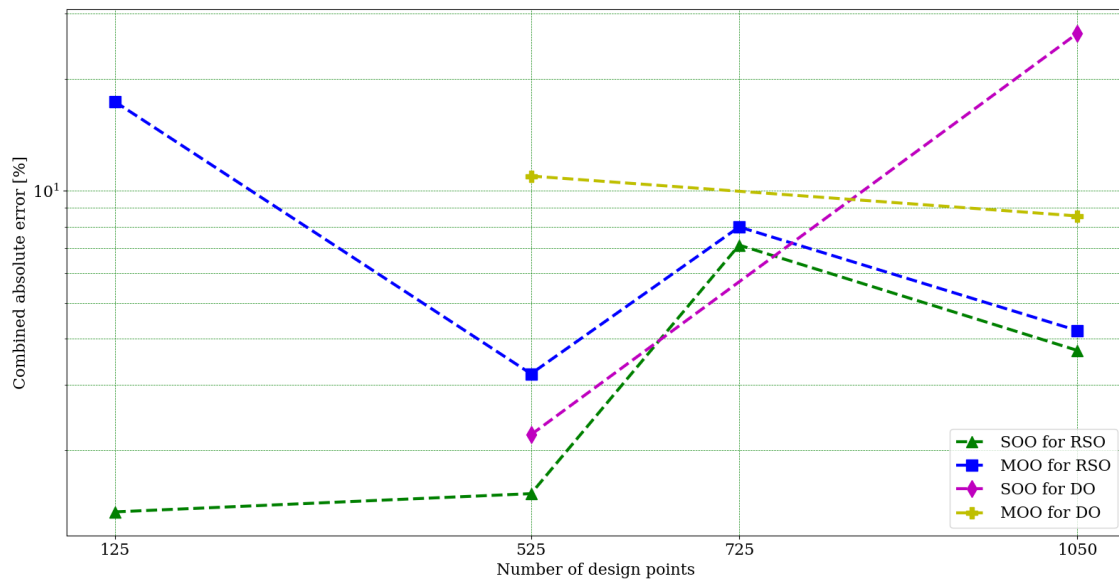


Figure 4.32 General comparison of the CAE calculated for all conducted experiments.

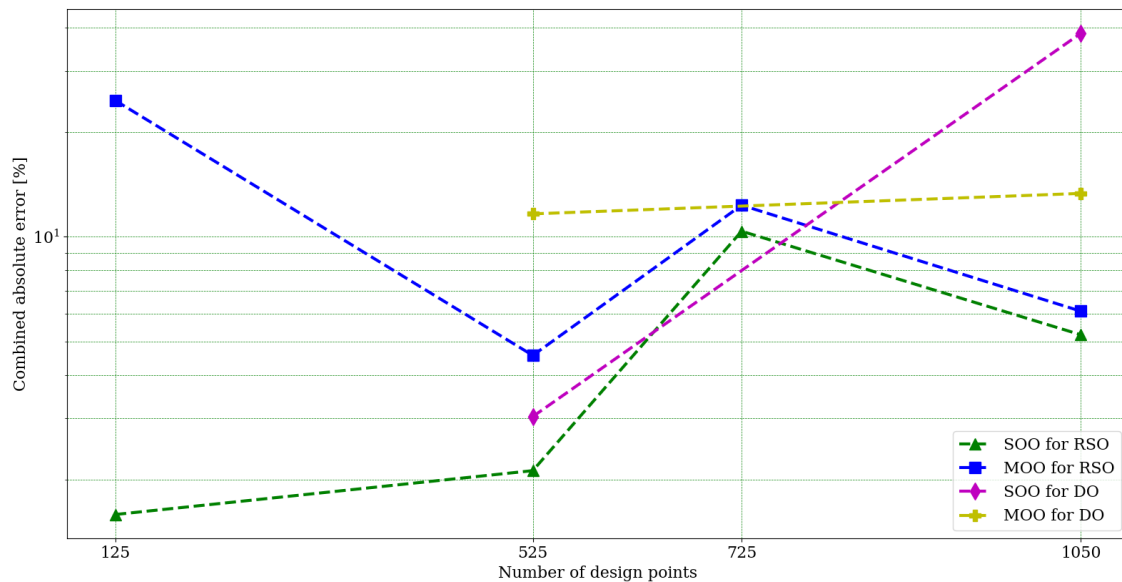


Figure 4.33 General comparison of the CAEYM calculated for all conducted experiments.

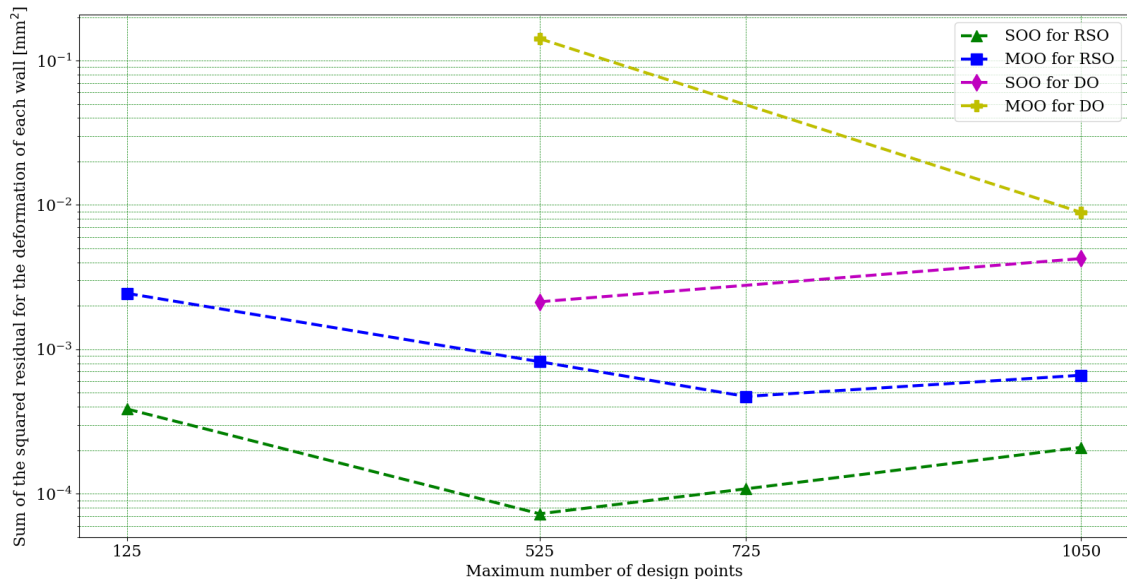


Figure 4.34 General comparison of the sum of the squared differences between the deformations in the updated model, calculated with the FE model, and those in the synthetic data.

those for DO across the various metrics used. However, for the case where 525 design points were used, RSO outperforms DO for SOO across all metrics, but, for MOO, RSO only outperforms DO in terms of the sum of the squared differences. In terms of CAE and CAEYM, SOO for DO using 525 design points presents better results than MOO for both RSO and MOO for the same number of design points. However, when analysing SOO for DO using 1050 design points, the results are completely opposite, as this stands out as the worst option both in terms of CAE and CAEYM. As previously mentioned, this unexpected behaviour may be a consequence of the increase in the number of design points leading to a lower global error on the RS that does not necessarily improve the RS's quality near the global optimal point, which corresponds to the wall thicknesses and Young Modulus used in the synthetic FE model.

Figure 4.34 also reveals that RSO outperforms DO for both SOO and MOO, regardless of the number of design points used. Nonetheless, an increase in the number of design points used seems to greatly benefit DO when MOO is use, which does not seem to be the case for RSO. Therefore, it would seem that a RS using 525 design points would be sufficient to approach satisfactory results. In fact, that RS would still guarantee better results than any of the experimented DO approaches, both for SOO and MOO.

All in all, RSO appears to be the best option for analysing the problem at hands, regardless of the number of design points used. Additionally, it does not seem beneficial to increase the number of design points in the RSO beyond 1050 design points, and 525 design points seems enough to guarantee good results for this particular case. Regarding DO, it would be interesting to evaluate its performance using a greater number of design points

and assess whether it can outperform RSO. Additionally, it would be relevant to re-conduct the optimization procedures several times in order to examine their behaviour and reach more comprehensive conclusions. From this analysis it would seem that SOO outperforms MOO for RSO. However, for the case of DO, SOO only outperforms MOO when using 525 design points. Moreover, MOO seems to improve in a more brisk manner than SOO for DO as the number of design points increases, so the aforementioned conclusion could be untrue if the number of design points increases significantly.

4.3.5 Conclusions

Additionally to the information presented before, the overall computational cost of running a RSO is slightly lower than that of running a DO, if the same number of FEM evaluations is imposed. Even though the FE model was re-evaluated a similar amount of times when creating the RS and when running the DO, the method for creating the RS is marginally lighter than the method used for the DO, given the stochastic nature of the latter. Furthermore, it is relevant to note that, although the optimization procedure used in the RSO ran a considerable amount of evaluations, it did so on the RS, which is simply a polynomial surface and is therefore computationally economical to analyse. Therefore, RSO seems more attractive for implementing in a DT as the FE model must only be reevaluated to create the RS, but not to run the optimization procedure, and therefore conduct model updating. All in all, even if the computational cost of conducting DO and RSO is similar for a single attempt at conducting model updating, the impact of reevaluating the FE model for each candidate solution evaluated in the optimization procedure is magnified as the number of model updating instances increases. That being said, when implementing an optimization tool for model updating in a DT, one must bear in mind that the optimization procedure will certainly be re-conducted numerous times, allowing for a precise follow-up of the structure's life-cycle. Therefore, the main focus must be on guaranteeing that the optimization procedure is light-weight and effective, thus creating an expeditious model updating tool. This is achieved when using RSO, which only requires consulting and running the FE model for creating the RS, not for running the optimization procedure used for model updating.

The results presented above make a strong case for the use of RSO, which, additionally to its competitive running time and computational cost, presents remarkably satisfactory results. Furthermore, for both RSO and DO, SOO seems to present better results on almost all aspects. Particularly, when combined with RSO, SOO presents the best results on all metrics, leading to the conclusion that RSO combined with SOO may be the best option when conducting model updating. However, if time and computational cost are not a restraint, then it might be interesting to opt for a DO approach using MOO with a large number of design points (larger than 1050 design points).

Chapter 5

A case study on model updating with experimental data

Based on the results and relevant conclusions taken from the case study on model updating with synthetic data, presented in previous chapter, it was possible to conduct a more complex case study, based on practical experiments, and explore other model updating techniques. The experimental setup consisted of a cube subject to internal pressure, similarly to what was done in the case study from chapter 4. *Ansys* was used to generate the simulation FEM model, but not to produce the RS nor conduct the optimization procedure. Alternatively, based on the data of a DOE matrix retrieved using the simulation model, a RS (metamodel) was created with *Python* scripting. Afterwards, PSO and GA were implemented to conduct optimization procedures, also with *Python* scripting, using the referred RS.

5.1 Problem specification

The problem at hands consists of a cube ($500 \times 500 \times 500$ mm) with two pipes, one on the side of the cube, for introducing pressure into the cube, and another one at the top of the cube, to install a pressure gauge. Two photographs of this cube can be found in figure 4.1, in chapter 4. This cube is made up of S235JR structural steel, which has a Young Modulus of about 210 GPa, and the walls are expected to be 6 mm thick, but are in fact somewhere between 5.4 and 6.6 mm thick, which is the dimensional tolerance for this type of element. The cube is pinned at three of the four corners of its base and the deformation of 4 walls (the top and bottom walls are not considered) was measured using 4 Linear Variable Differential Transformers (LVDT). Pressure was gradually introduced into the cube until it reached 2 bar (200 kPa) and relevant data was collected, as is explained in section 5.2.

The material considered for the FE model is very similar to the S235JR structural steel, except it has a Young Modulus of 210 GPa, and the walls are defined as being 6 mm thick.

The maximum deformation of each of the considered walls is evaluated for various pressure values, determined according to the pressure introduced into the real cube. Afterwards, using the DOE, a data set including various possible combinations of wall thicknesses, wall deformations and Young Modulus is generated in *Ansys*. Using said data set, a RS is created in Python, and two different model updating approaches (GA and PSO) are implemented to determine the thickness of the four walls and the Young Modulus. Since the case study from chapter 4 showed that a combination of RSO with a SOO approach is the most accurate, this type of approach was used for the model updating procedure in the present study.

5.2 Experimental set up and data retrieval

In order to conduct the experimental analysis, the cube was set on a table with supports on 3 of its base corners (front left corner and back right and left corners). Additionally, 4 LVDTs were installed at the center of each of the 4 walls, as can be seen in figure 5.1. The data retrieved from these sensors was analysed in *LabView* and the signal was processed using a second-order Butterworth low-pass filter with a cut-off frequency of 5 Hz. This filter removes signal oscillations with a frequency higher than 5 Hz, which are considered noise. This noise may be due to any minor vibration in the table where the cube was set, electromagnetic noise that affects the signal sent through the cables, oscillations in voltage provided by the source or measurement uncertainty in the LVDTs. However, it is relevant to note that none of this was significantly observed in the raw signals of this experiment, so the differences due to the filter application are almost nonexistent.

As previously mentioned, pressure was gradually introduced into the cube until approximately 2 bar (200 kPa) were reached. In order to facilitate retrieving and processing data, pressure was introduced at increments of around 0.25 bar (25 kPa) and the corresponding deformation for each of those pressure levels was recorded. Notice that it was not possible to strictly abide by the aforementioned protocol due to limitations imposed by the compressed air system used to introduce pressure into the cube. Table 5.1 presents the pressure levels analysed during the experimental trial. It is relevant to note that the first and last pressure levels correspond to the moments prior and after introducing pressure, which indicates that the pressure gauge might have an offset of 0.8 kPa. That being said, table 5.2 presents the correct pressures levels, where the pressure offset was subtracted to the recorded pressure levels, and the average deformation value for each wall at each pressure level. Figure 5.3 presents the evolution of the deformation of each wall as a function of the applied internal pressure obtained from the experimental data.



Figure 5.1 Experimental setup: pressure gauge and LVDT sensors.

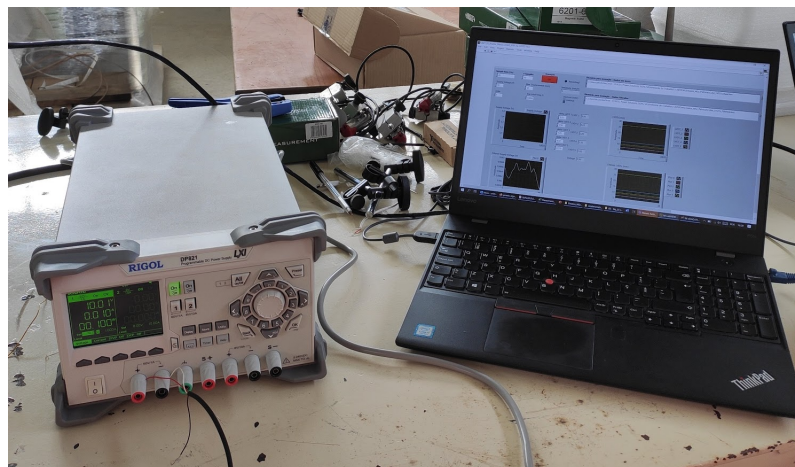


Figure 5.2 Experimental setup: power supply and *LabView*.

Table 5.1 Pressure levels and wall deformations for the experimental trial.

Pressure (kPa)
0.8
29.8
50.6
74.3
95.6
128.8
150.5
181.1
198.3
0.9

Table 5.2 Pressure levels and wall deformations for the experimental trial.

Pressure (kPa)	Deformation, δ (mm)			
	Front Wall	Back Wall	Right Wall	Left Wall
0.0	0.0000	0.0000	0.0000	0.0000
29.0	0.6973	0.7114	0.8078	0.7028
49.8	1.2482	1.3066	1.4275	1.2084
73.5	1.8974	1.9574	2.0871	1.7423
94.8	2.4474	2.5038	2.6628	2.2244
128.0	3.1536	3.0740	3.4410	2.8708
149.7	3.6701	3.5517	3.9381	3.3083
180.3	4.4353	4.2084	4.6741	3.9515
197.5	4.8228	4.5684	5.0401	4.2334
0.1	0.0003	0.0914	0.1105	0.0925

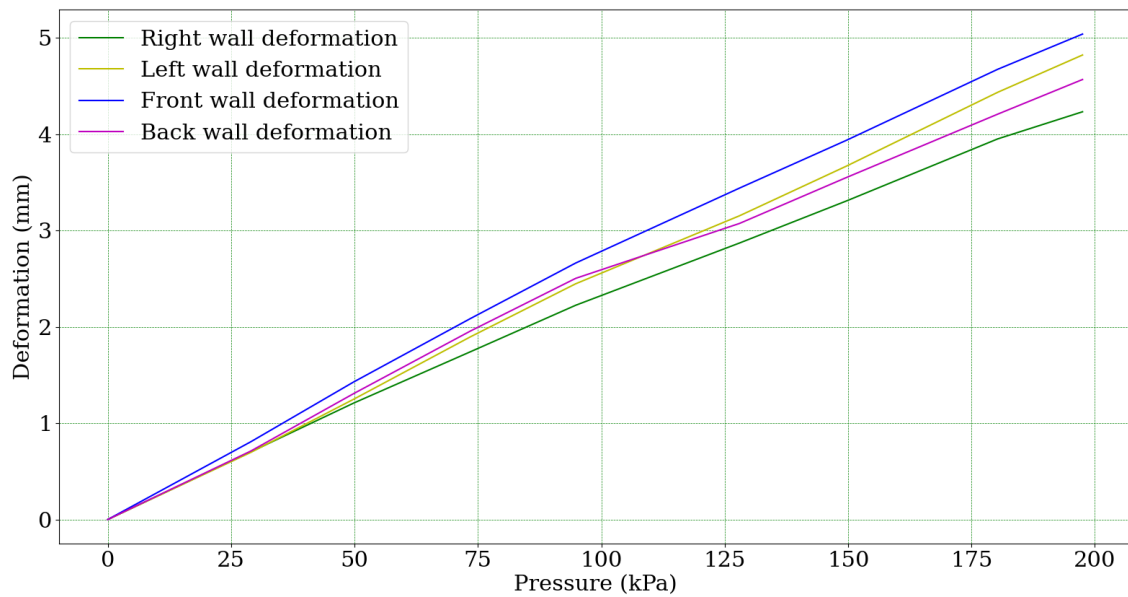


Figure 5.3 Evolution of the deformation of each wall as a function of the applied internal pressure obtained from the experimental data.

5.3 Ansys: model preparation

In order to conduct the model updating procedure, 8 models were created in the same *Workbench*. All these models are similar, except for the applied pressure, which was set to the values presented in table 5.2. Notice that the first and last pressure levels were not considered for these models. Each model was named after the pressure level it was subjected to, as can be seen in figure 5.4.

The aforementioned models were developed using *SpaceClaim*, a CAD tool embedded in *Ansys*. In order to simplify the analysis, all surfaces were converted to *midsurfaces* during geometry preparation (before meshing), allowing the use of shell elements when the mesh for the FE analysis was created. Notice that this approximation is valid since shell elements are used to model structures in which one dimension, in this case the thickness, is significantly smaller than the other dimensions, [38].

After developing the model in *SpaceClaim*, the FE mesh was generated in *Mechanical*, a structural FEA tool. These tools can be accessed through the *Workbench*, which is *Ansys*'s main page. The created mesh is made up of quadrilateral and triangular elements, with greater abundance of the first type. The size of these elements varies according to their location, with a greater density of elements near potentially problematic areas, such as holes or any other area that may be prone to stress concentration.

Afterwards, the boundary conditions and loading scheme were defined. As mentioned before, three of the four vertices of the base of the cube were pinned (the front left vertex and the back right and left vertices), as can be seen in figure 5.6, and all the welded and bolted connections were considered as bounded connections. This is an approximation

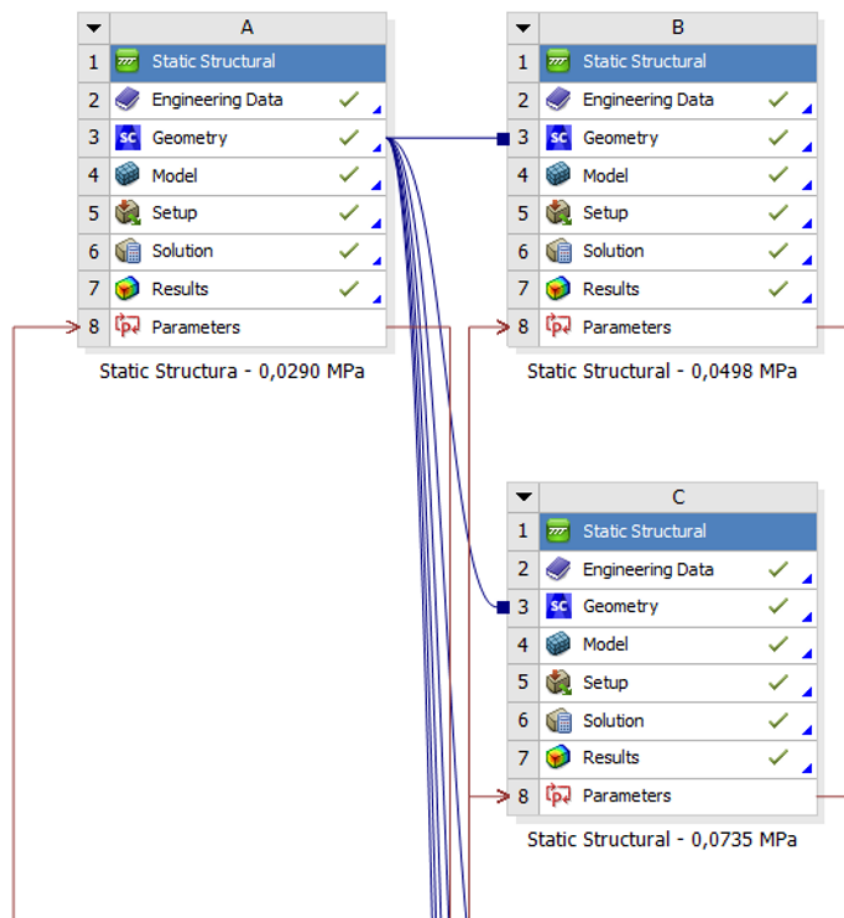


Figure 5.4 *Ansys Workbench* blocks for the models at various pressure levels.

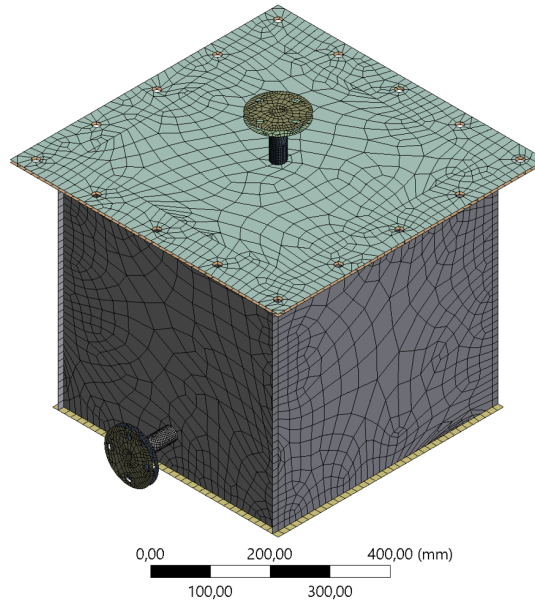


Figure 5.5 FE mesh generated in *Ansys*.

which minimizes computing time, though it may affect the accuracy of the obtained results. After defining these conditions, the internal pressure was applied (according to the corresponding model) to all 6 walls of the cube, as can be seen in figure 5.7.

5.3.1 Defining parameters

The aforementioned models were considered to be connected in the *Workbench* and for each of them the thickness and deformation of each of the 4 walls (Front, Back, Left and Right) were set as parameters, as well as the Young Modulus. Since the Young Modulus and thickness values for each wall are independent of the applied pressure, these parameters were set to be equal between all models. For example, the thickness of the right wall is the same for the model at 29.0 kPa, the model at 49.8 kPa, the model at 73.5 kPa and so on. This relationship between parameters is defined in the parameter set, as can be seen in figure 5.10.

5.3.2 Creating the RS

As previously mentioned, the RS was created using data retrieved from *Ansys*. Therefore, using the aforementioned 8 models (each corresponding to a different pressure level), the DOE was created. The data retrieved from the DOE, which relates the various parameters considered in this experiment, was then processed through a *Python* script to create a RS, using an interpolation algorithm.

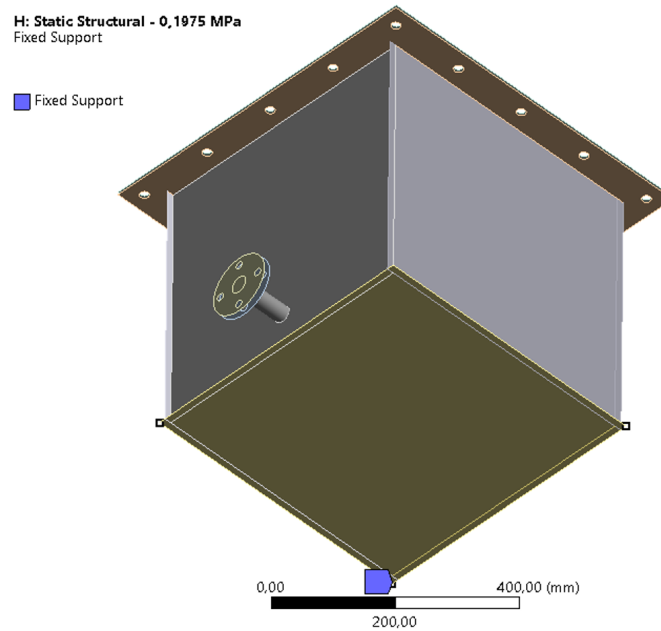


Figure 5.6 Fixed supports applied to the FE model of the cube.

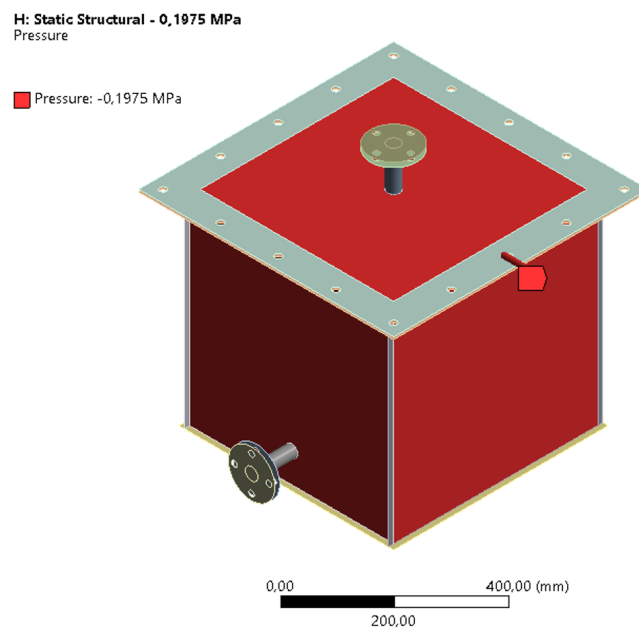


Figure 5.7 Internal pressure applied to the 6 walls of one of the FE models of the cube (model subject to an internal pressure of 197.5 kPa).

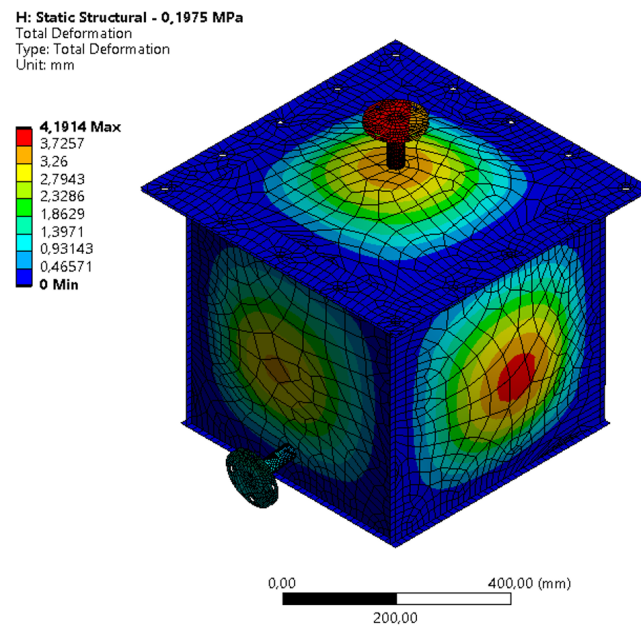


Figure 5.8 Total deformation, calculated in *Ansys*, for the model subject to an internal pressure of 197.5 kPa.

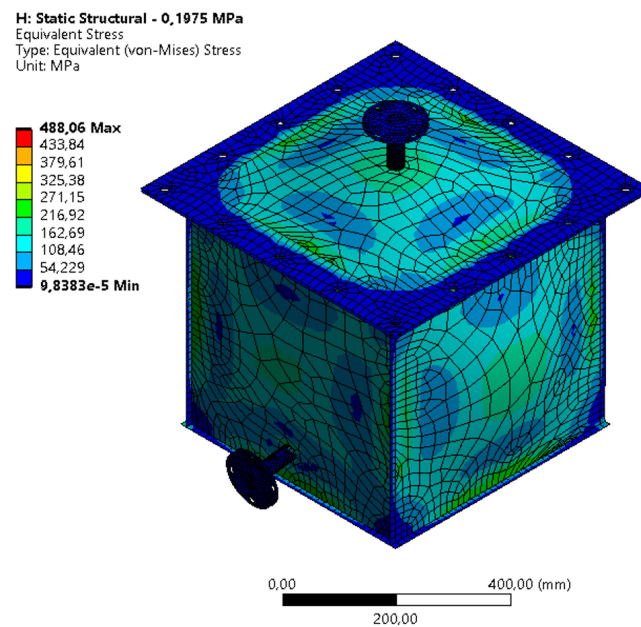


Figure 5.9 Von Mises equivalent stress, calculated in *Ansys*, for the model subject to an internal pressure of 197.5 kPa.

ID	Parameter Name	Value	Unit
[-] Input Parameters			
[-] Static Structural - 0,0290 MPa (A1)			
P8	Midsurface - N111000080120\Midsurface 1 Thickness	6	mm
P15	Midsurface - N111000080130\Midsurface 1 Thickness	6	mm
P18	Midsurface - N111000080111\Midsurface 1 Thickness	6	mm
P19	Midsurface - N111000080130\Midsurface 1 Thickness	6	mm
P94	Young's Modulus	2,099E+11	Pa
[-] Static Structural - 0,0498 MPa (B1)			
P31	Midsurface - N111000080120\Midsurface 1 Thickness	6	mm
P32	Midsurface - N111000080130\Midsurface 1 Thickness	6	mm
P33	Midsurface - N111000080111\Midsurface 1 Thickness	6	mm
P34	Midsurface - N111000080130\Midsurface 1 Thickness	6	mm
P95	Young's Modulus	2,099E+05	MPa
Property		Value	
[-] General			
Expression	P8		
Usage	Input		
Description			
Error Message			
Expression Type	Derived		
Quantity Name	Length		

Figure 5.10 Back wall thickness parameter being defined as equal between models in the *Parameter Set*.

Using *Python* scripting for creating the RS allows for a greater degree of independence from *Ansys*, which is an attractive feature, particularly in a corporate environment, as working with licensed software may not always be possible and available at all times. Additionally, having created the RS in *Python* based on the original model, it can then be used to follow the entire life-cycle of the product, as the model parameters considered in the RS may be updated as many times as necessary. This further emphasizes the lasting independence from licensed software. All in all, creating the RS and running the optimization procedure in *Python* is a step forward towards creating a DT which can be used and updated continuously, using for example data measured in-field, during the in-service phase of the product.

5.3.2.1 Creating the DOE

In the DOE, the upper and lower bounds for each input parameter were set, as shown in table 5.3. The method for creating the design points was 'Custom', which allowed for the customization of the desired design points. That being said, in order to create a representative, yet reasonably sized, set of initial design points, the lower and upper bounds, plus two intermediate values for each input parameter, were used to create these design points. This information is also presented in table 5.3. Therefore, for each parameter, 4 possible values were evaluated, which generated $4^5 = 1024$ initial design points (each design point represents a possible combination between the various values evaluated for each input parameter). Notice that, although 5 input parameters were set for each model (the thickness of each of the 4 walls and the Young Modulus), making up a total of 40 input parameters, these parameters were set to be equal between all models. Therefore, in terms of the DOE, only 5 input parameters are relevant.

Table 5.3 Upper and lower bounds and intermediate values considered for the input parameters.

Parameter	Unit	Lower Bound	Intermediate Value		Upper Bound
			Lower	Upper	
$t_{FrontWall}$	mm	5.4	5.8	6.2	6.6
$t_{BackWall}$	mm	5.4	5.8	6.2	6.6
$t_{RightWall}$	mm	5.4	5.8	6.2	6.6
$t_{LeftWall}$	mm	5.4	5.8	6.2	6.6
Young Modulus	GPa	180	193	206	220

After establishing the parameters for the DOE, *Ansys* simulated the corresponding deformation values for each design point, therefore completing the data set. Having done that, it was possible to extract said data set as a CSV document.

It is relevant to point out that the scheme used for creating the DOE is computationally demanding and particularly time consuming. However, an interesting alternative would

be to create an individual *Workbench* file for each pressure level and create the DOE for each of these models separately, but using the same order of design points. This way, it is possible to guarantee the analysis of the same combinations of wall thicknesses and Young Modulus (which is to say that each design point is analysed across all pressure levels) while also being able to generate all DOE data sets at the same time by running the DOE for all models in parallel. After obtaining each data set, they can all be combined in a single Excel file to be later on read by Python to create the response surface and run the optimization procedures.

5.3.2.2 Interpolating the RS design points

After exporting the data set generated in *Ansys* as a CSV document, it was possible to import said information into Python. To do that, various modules were imported into Python, such as:

- `os` : presents miscellaneous operating system interfaces;
- `pandas` : provides open source data analysis and manipulation tools;
- `openpyxl` : allows one to read or write Excel `xlsx/xlsm/xltx/xltm` files.

After reading the data set and introducing it into Python as lists (each parameter's values were placed into a list), it was possible to create the RS. Since the main focus is not necessarily on obtaining the equation that governs the RS but instead on having a function which allows the optimization algorithm to evaluate the fitness of each candidate solution, an interpolation algorithm was chosen for this purpose. That being said, various interpolation algorithms are available to be used in Python, namely in the SciPy library. The interpolation algorithms found in this library are:

- 1D interpolation (`interp1d`)
- 2D interpolation (`interp2d`)
- Multivariate data interpolation (`griddata`)
- Spline interpolation
 - Spline interpolation in 1D: Procedural (`interpolate.splXXX`)
 - Spline interpolation in 1D: Object-oriented (`UnivariateSpline`)
 - 2D spline representation: Procedural (`bisplrep`)
 - 2D spline representation: Object-oriented (`BivariateSpline`)
- Radial basis function (`Rbf`)

Given the fact that the intended interpolation has 5 input variables (wall thicknesses and Young Modulus) and 32 output variables (wall deformations of the 4 walls for the 8 pressure levels), only multivariate data interpolation (MDI) and radial basis function (RBF) seem to fit the requirements. That being said, both algorithms were tested to assess their performance.

Radial basis function (RBF) The RBF is based on a sophisticated approximation theory approach for building high-order accurate interpolants of unstructured data, potentially in high-dimensional spaces. A RBF is a function with points specified as distances from a center and the interpolant is implemented as a weighted sum of RBFs. If said center coincides with the origin, then one can say that the RBF is symmetrical around the origin, [40].

The Bell Curve is an example of an RBF where points are represented as number of standard deviations from the mean. Formally, an RBF may be defined as a function that can be written as, [40]:

$$f(x) = f(||x||) \quad (5.1)$$

Based on this theoretical ground, a RBF is a scalar valued function in n -dimensional space whose value at x can be expressed in terms of $r = ||x - c||$, where c is the center of the RBF. Python's SciPy RBF interpolant class resorts to RBFs to interpolate n -dimensional data that is scattered and potentially noisy. Therefore, the RBF interpolant for the scalar valued observations $\mathbf{d} = [d_1, \dots, d_n]^T$, made at scattered locations x_1, \dots, x_n , is a linear combination of RBFs centered at x plus a polynomial with a specified degree. The RBF interpolant is parameterized as follows

$$f(x) = \sum_{i=1}^n a_i \phi(||x - x_i||) + \sum_{j=1}^m b_j p_j(x) \quad (5.2)$$

where ϕ is an RBF and $p_1(x), \dots, p_m(x)$ are monomials that span the space of polynomials with a specified degree. The coefficients $\mathbf{a} = [a_1, \dots, a_n]^T$ and $\mathbf{b} = [b_1, \dots, b_m]^T$ are the solutions to the linear equations

$$(\mathbf{K} + \sigma^2 \mathbf{I})\mathbf{a} + \mathbf{P}\mathbf{b} = \mathbf{d} \quad (5.3)$$

and

$$\mathbf{P}^T \mathbf{a} = \mathbf{0} \quad (5.4)$$

where \mathbf{K} is a matrix of RBFs with components $K_{ij} = \phi(||x_i - x_j||)$, \mathbf{P} is a matrix with components $P_{ij} = p_j(x_i)$, and σ is a smoothing parameter that regulates how well the observations should be fit. When σ is zero, the observations are perfectly fit, [41, 42, 43, 44].

The solution for \mathbf{a} and \mathbf{b} is unique if the chosen RBF is positive definite and \mathbf{P} has full column rank. If the selected RBF is conditionally positive definite of order q and \mathbf{P} has full column rank, the solution is unique provided that the degree of the monomial terms is at least $q - 1$, [41, 42, 43, 44].

Currently, SciPy offers two RBF interpolant classes: `scipy.interpolate.Rbf` and `scipy.interpolate.RBFInterpolator`. The first class is said to be a legacy code, which refers to code that was developed a long time ago, as well as contemporary code that was written without tests for some reason. Nonetheless, both classes are analysed here.

In order to properly call the RBF interpolant class `scipy.interpolate.Rbf`, various parameters can be defined, namely:

- arguments (in the form of arrays): x, y, z, \dots, d , where x, y, z, \dots are the coordinates of the nodes and d is the array of values at the nodes.
- function (optional): the RBF, based on the radius, r , given by the norm (Euclidean distance). The default is ‘multiquadric’, but the following RBFs are available:

- ‘multiquadric’: $\phi(r) = \sqrt{1 + (\varepsilon r)^2}$

- ‘inverse’: $\phi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$

- ‘gaussian’: $\phi(r) = e^{-(\varepsilon r)^2}$

- ‘linear’: $\phi(r) = r$

- ‘cubic’: $\phi(r) = r^3$

- ‘quintic’: $\phi(r) = r^5$

- ‘thin_plate’: $\phi(r) = r^2 \log(r)$

- ε (optional): adjustable constant for gaussian, inverse and multiquadric functions - defaults to the average distance between nodes.
- smooth (optional): refers to the smoothing parameter, σ . Values greater than zero increase the smoothness of the approximation. If set to 0, which is the default, the function will go through all the nodal points, creating a true interpolation of the evaluated points.
- neighbours, K (optional): if specified, the value of the interpolant at each evaluation point will be computed using only the specified number of nearest data points, K . All the data points are used by default.

Notice that only the arguments are mandatory, while the remaining parameters are optional. When no option is chosen for the latter parameters, the default option is selected by the algorithm.

Having defined the arguments, the RBF interpolant `scipy.interpolate.Rbf` is created as follows:

Listing 5.1 Defining the RBF interpolation function for `scipy.interpolate.Rbf`

```
d_right_290_interp = scipy.interpolate.Rbf(t_front_DOE,
                                           t_back_DOE, t_right_DOE, t_left_DOE,
                                           young_modulus_DOE, d_right_290_DOE,
                                           function='multiquadric')
```

where:

- 'd_right_290_interp' is the callable interpolation function. In this case, the deformation of the right wall when subject to an internal pressure of 29.0 kPa is being defined.
- 't_front_DOE', 't_back_DOE', 't_right_DOE', 't_left_DOE' and 'young_modulus_DOE' are the lists containing the values for the wall thicknesses and Young Modulus for each point in the DOE data set.
- 'd_right_290_DOE' is the array of the data point values corresponding to the data point coordinates for each point in the DOE data set. In this case, these values correspond to the deformation of the right wall when subject to an internal pressure of 29.0 kPa.
- function is the function defined for the RBF interpolant. In this case, 'multiquadric' was chosen.

In order to evaluate the interpolation at a given point (calculate the interpolation of the deformation for a given set of wall thicknesses and Young Modulus), the following code line must be written:

Listing 5.2 Calling the RBF interpolation function for `scipy.interpolate.Rbf`

```
d_right_290 = d_right_290_interp(t_front, t_back, t_right,
                                 t_left, young_modulus)
```

where:

- 'd_right_290_interp' is the callable interpolation function. In this case, it's the interpolation function for the deformation of the right wall when subject to an internal pressure of 29.0 kPa.
- 't_front', 't_back', 't_right', 't_left' and 'young_modulus' are the values for the wall thicknesses and Young Modulus at the point at which the deformation must be interpolated.
- 'd_right_290' is the interpolated value for the deformation of the right wall when subject to an internal pressure of 29.0 kPa calculated at the point with wall thicknesses 't_front', 't_back', 't_right', 't_left' and Young Modulus 'young_modulus'.

Alternatively, if one intends to call the RBF interpolant class *scipy.interpolate.RBFInterpolator*, the following parameters can be defined:

- y (in the form of a single array): array of the data point coordinates.
- d (in the form of an array): array of values at points y .
- kernel (optional): the type of RBF. The default is ‘thin_plate_spline’, but the following RBFs are available:
 - ‘multiquadric’: $\phi(r) = -\sqrt{1 + (\varepsilon r)^2}$
 - ‘inverse_multiquadric’: $\phi(r) = \frac{1}{\sqrt{1 + (\varepsilon r)^2}}$
 - ‘inverse_quadratic’: $\phi(r) = \frac{1}{1 + (\varepsilon r)^2}$
 - ‘gaussian’: $\phi(r) = e^{-(\varepsilon r)^2}$
 - ‘linear’: $\phi(r) = -r$
 - ‘cubic’: $\phi(r) = r^3$
 - ‘quintic’: $\phi(r) = -r^5$
 - ‘thin_plate_spline’: $\phi(r) = r^2 \log(r)$
- ε (optional): shape parameter that scales the RBF input. If the kernel is ‘linear,’ ‘thin plate spline,’ ‘cubic,’ or ‘quintic,’ this value is 1 and may be disregarded because it has the same effect as scaling the smoothing parameter. This must be indicated otherwise.
- smoothing (optional): refers to the smoothing parameter, σ . Values greater than zero increase the smoothness of the approximation. If set to 0, which is the default, the function will go through all the nodal points, creating a true interpolation of the evaluated points.
- neighbours (optional): if specified, the value of the interpolant at each evaluation point will be computed using only the specified number of nearest data points (neighbours). All the data points are used by default.

Notice that only arguments y and d are mandatory while the remaining parameters are optional. When no option is chosen for the latter parameters, a default option is selected by the algorithm.

Having defined the arguments, the RBF interpolant *scipy.interpolate.RBFInterpolator* is defined as follows:

Listing 5.3 Defining the RBF interpolation function for *scipy.interpolate.RBFInterpolator*

```
d_right_290_interp = scipy.interpolate.RBFInterpolator(
    data_points, d_right_290_DOE,
    kernel='thin\_plate\_spline ', epsilon=1)
```


where:

- 'd_right_290_interp' is the callable interpolation function. In this case, the deformation of the right wall when subject to an internal pressure of 29.0 kPa is being defined.
- 'data_points' is the array of the data points coordinates, created from the lists 't_front_DOE', 't_back_DOE', 't_right_DOE', 't_left_DOE' and 'young_modulus_DOE' which contain the values for the wall thicknesses and Young Modulus for each point in the DOE data set.
- 'd_right_290_DOE' is the array of the data point values corresponding to the data point coordinates for each point in the DOE data set. In this case, these values correspond to the deformation of the right wall when subject to an internal pressure of 29.0 kPa.
- kernel is the function defined for the RBF interpolant. In this case, 'thin_plate_spline' was chosen.
- epsilon is the shape parameter defined for the RBF interpolant. In this case, the value 1 was chosen for this parameter.

In order to evaluate the interpolation at a given point (calculate the interpolation of the deformation for a given set of wall thicknesses and Young Modulus), the following code line must be written:

Listing 5.4 Calling the RBF interpolation function for `scipy.interpolate.RBFInterpolator`

```
d_right_290 = d_right_290_interp(evaluation_points)
```

where:

- 'd_right_290_interp' is the callable interpolation function. In this case, it's the interpolation function for the deformation of the right wall when subject to an internal pressure of 29.0 kPa.
- 'evaluation_point' is the array containing the coordinates of the point where the interpolation function must be evaluated. This means that this array contains the values for the wall thicknesses and Young Modulus at the point at which the deformation must be interpolated.
- 'd_right_290' is the interpolated value for the deformation of the right wall when subject to an internal pressure of 29.0 kPa calculated at the point 'evaluation_point'.

Multivariate data interpolation (MDI) In order to properly call the MDI interpolant class *scipy.interpolate.griddata*, various parameters can be defined, namely:

- points : array of the data point coordinates.
- values (in the form of an array): array of values at the points.
- x_i (in the form of a single array): array containing the points at which to interpolate the data set.
- method (optional): method of interpolation. The default is ‘nearest’, but the following methods are available:
 - ‘nearest’: returns the value at the data point closest to the point of interpolation.
 - ‘linear’: the interpolant is built by triangulating the input data and conducting linear barycentric interpolation on each triangle.
 - ‘cubic (1-D)’: returns the value determined from a cubic spline.
 - ‘cubic (2-D)’: the interpolant is built by triangulating the input data and applying a piecewise cubic interpolating Bezier polynomial to each triangle. The interpolant is guaranteed to be continuously differentiable. The interpolant gradients are set in such a way that the curvature of the interpolating surface is approximatively minimized.

Notice that only the points and values are mandatory while the remaining parameters are optional. When no option is chosen for the latter parameters, the default option is selected by the algorithm.

Having defined the arguments, the MDI interpolant class *scipy.interpolate.griddata* can be called as is shown in listing 5.5. Notice that, contrary to both RBF classes, *griddata* directly calculates the interpolated values at the desired point, thus avoiding the need to define an interpolation function and calling it later on.

Listing 5.5 Calling the MDI interpolant class - *scipy.interpolate.griddata*

```
d_right_290 = scipy.interpolate.griddata(data_points ,
                                         d_right_290_DOE, evaluation_points ,
                                         method='linear ')
```

where:

- ‘d_right_290’ is the interpolated value for the deformation of the right wall when subject to an internal pressure of 29.0 kPa calculated at the point ‘evaluation_point’.

- 'data_points' is the array of the data points coordinates, created from the lists 't_front_DOE', 't_back_DOE', 't_right_DOE', 't_left_DOE' and 'young_modulus_DOE' which contain the values for the wall thicknesses and Young Modulus for each point in the DOE data set.
- 'd_right_290_DOE' is the array of the data point values corresponding to the data point coordinates for each point in the DOE data set. In this case, these values correspond to the deformation of the right wall when subject to an internal pressure of 29.0 kPa.
- 'method' is the method defined for the MDI interpolant. In this case, 'linear' was chosen.
- 'evaluation_point' is the array containing the coordinates of the point where the interpolation function must be evaluated. This means that this array contains the values for the wall thicknesses and Young Modulus at the point at which the deformation must be interpolated.

5.3.2.3 Interpolation function and RS validation

In order to choose the most appropriate class between the RBF interpolant classes *scipy.interpolate.Rbf* and *scipy.interpolate.RBFInterpolator* and the MDI interpolant class *scipy.interpolate.griddata*, various verification points, created using the FE model, were used to evaluate the relative error found in the interpolation provided by functions from each candidate class. With the purpose of creating an extensive set of verification points, an approach similar to the one used to create the DOE data set was conducted, allowing for an expeditious analysis of the FE model's behaviour for different combinations of wall thicknesses and Young Modulus. Here, three possible thicknesses were considered for each wall, as well as three possible Young Modulus values, as is presented in table 5.4. These values were selected from the existing intervals between the values used to generate the DOE data set, presented in table 5.3, and all possible combinations of these values were considered, leading to a total of $3^5 = 243$ validation points.

Table 5.4 Young Modulus and wall thickness values considered for the validation points.

Parameter	Unit	Value 1	Value 2	Value 3
$t_{FrontWall}$	mm	5.6	6	6.4
$t_{BackWall}$	mm	5.6	6	6.4
$t_{RightWall}$	mm	5.6	6	6.4
$t_{LeftWall}$	mm	5.6	6	6.4
Young Modulus	GPa	185	193	215

Each validation point was interpolated using each of the functions available for each of the considered classes and the results were analysed using Excel. Notice that all the

optional parameters available in each class were set as default for all analysed functions. For each candidate point analysed using each class and function, a relative error was calculated between the deformation obtained at each pressure level according to the interpolation and the deformation obtained according to the FE model, for the same pressure level. Equation 5.5 was used to calculate this relative error.

$$Relative\ Error\ (\%) = \frac{\delta_{Interpolation} - \delta_{FEModel}}{\delta_{FEModel}} \quad (5.5)$$

To facilitate the comparison between the various options, the maximum and average of the absolute values of the relative errors (Max. Rel. Error (%) and Avg. Rel. Error (%), respectively) were calculated for each interpolation and the results are presented in tables 5.5, 5.6 and 5.7. Notice that for MDI only the 'nearest' and 'linear' methods were considered given the fact that the remaining ones ('cubic 1-D' and 'cubic 2-D') cannot be used to interpolate functions with 5 dimensions.

Table 5.5 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available function in RBF interpolant class *scipy.interpolate.Rbf*.

Function	Max. Rel. Error (%)	Avg. Rel. Error (%)
Multiquadric	49.9	13.0
Inverse	56.2	45.1
Gaussian	100	100
Linear	40.9	11.6
Cubic	81.6	23.1
Quintic	566	248
Thin plate	66.3	15.7

Table 5.6 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available kernel in RBF interpolant class *scipy.interpolate.RBFInterpolator*.

Kernel	Max. Rel. Error (%)	Avg. Rel. Error (%)
Multiquadric	63.8	19.3
Inverse Multiquadric	46.0	18.0
Inverse Quadratic	47.2	17.6
Gaussian	49.3	17.8
Linear	44.2	12.7
Cubic	70.1	11.8
Quintic	52.9	4.85
Thin Plate	66.9	15.8

Based on the aforementioned tables, MDI using the 'linear' method appears to be the best option for the present study, displaying the lowest average relative error and the second lowest maximum relative error. However, for the majority of the analysed points interpolated using this method, the relative error found was below 1%, and errors greater

Table 5.7 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available method in the MDI interpolant class *scipy.interpolate.griddata*.

Method	Max. Rel. Error (%)	Avg. Rel. Error (%)
Nearest	67.7	11.2
Linear	45.7	2.30

than that were only found for pressure levels of 149.7 kPa and 197.5 kPa. In fact, a similar discrepancy between the maximum errors for these two pressure levels and the maximum errors for the remaining pressure levels was detected for all analysed interpolant classes and functions. This was found to be due to an error in the data set of the validation points, which only affected those two pressure levels. Therefore, in the interest of time, the maximum and average relative errors for each interpolant class and function were calculated disregarding those two pressure levels. The results are presented in tables 5.8, 5.9 and 5.10.

Table 5.8 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available function in RBF interpolant class *scipy.interpolate.Rbf*, disregarding pressure levels 149.7 kPa and 197.5 kPa.

Function	Max. Rel. Error (%)	Avg. Rel. Error (%)
Multiquadric	27.6	11.4
Inverse	56.2	45.7
Gaussian	100	100
Linear	24.7	10.6
Cubic	43.4	21.0
Quintic	478	235
Thin plate	20.7	13.8

Table 5.9 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available kernel in RBF interpolant class *scipy.interpolate.RBFInterpolator*, disregarding pressure levels 149.7 kPa and 197.5 kPa.

Kernel	Max. Rel. Error (%)	Avg. Rel. Error (%)
Multiquadric	30.7	17.4
Inverse Multiquadric	41.3	17.4
Inverse Quadtratic	43.1	17.2
Gaussian	45.3	17.4
Linear	24.9	11.6
Cubic	27.0	9.87
Quintic	7.99	3.13
Thin Plate	20.6	14.0

Table 5.10 Maximum and average of the absolute values of the relative errors obtained from the interpolations using each available method in the MDI interpolant class *scipy.interpolate.griddata*, disregarding pressure levels 149.7 kPa and 197.5 kPa.

Method	Max. Rel. Error (%)	Avg. Rel. Error (%)
Nearest	15.9	9.8
Linear	1.64	0.770

Based on tables 5.8, 5.9 and 5.10, one can state that MDI using the 'linear' method seems to be the best option for the present study, displaying the lowest maximum and average relative error. One should bear in mind, however, that the pressure levels which were disregarded are two of the highest pressure levels and the errors found for those are expected to be higher than the errors found for lower pressure levels, even if marginally. Nonetheless, this does not alter the conclusions drawn from the present study.

For most of the analysed interpolation options, the maximum relative error presented in tables 5.8 and 5.9 is surprisingly high compared to the average relative error, which indicates that those interpolations may not be smooth and may present troublesome areas. For that reason, a different analysis was conducted which consisted of representing the evolution of the deformation of a given wall as a function of that same wall's thickness and as a function of the Young Modulus, allowing for a more comprehensive representation of the potential RSs created with the aforementioned interpolation algorithms. Through that analysis, an interesting aspect regarding the RBF interpolation classes was discovered. Considering both RBF interpolant classes and respective available functions or kernels, the evolution of the interpolated value of the right wall's deformation as a function of that same wall's thickness for an internal pressure of 197.5 kPa is presented in figures 5.11 and 5.12, and the evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus is presented in figures 5.13 and 5.14. Notice that the thickness of each of the remaining three walls was set to 6 mm and the Young Modulus was set to 210 GPa for the first graph, while for the second graph all walls were considered to be 6 mm thick. Despite only presenting the behaviour of the interpolation functions for the right wall, a similar behaviour was found for the remaining three walls. Even if a more comprehensive analysis could be conducted using 3D graphs, these 2D graphs provide an insight into the shape of the RSs, in the most simple and effective way. The fact that more than 3 variables are considered in the present analysis is a hurdle when trying to represent the RS, hampering the task of validating it's quality.

Based on the aforementioned graphs, the shape of the RS as a function of the thickness of each wall appears to be particularly smooth, leading to the conclusion that the deformation as a function of the wall thickness is correctly interpolated for most of the functions/kernels used. Notice, however, that, for *scipy.interpolate.Rbf*, functions 'quintic', 'inverse' and 'gaussian' present terrible approximations, which was expected based on the maximum and average relative errors analysed previously. For *scipy.interpolate.RBFInterpolator*, on

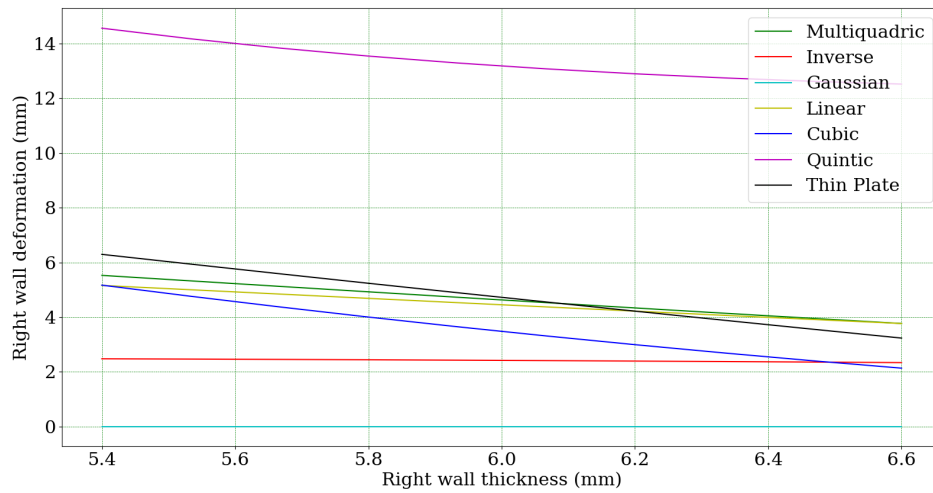


Figure 5.11 Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available function in RBF interpolant class *scipy.interpolate.Rbf*.

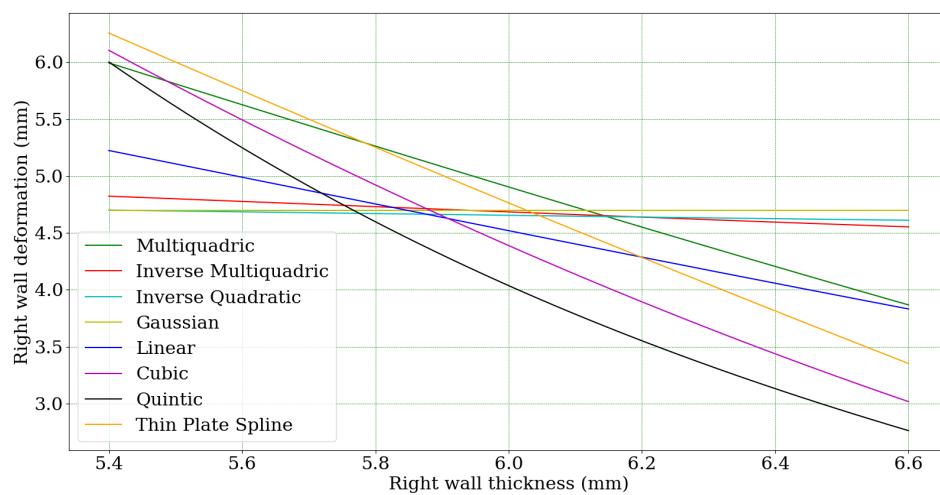


Figure 5.12 Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available kernel in RBF interpolant class *scipy.interpolate.RBF Interpolator*.

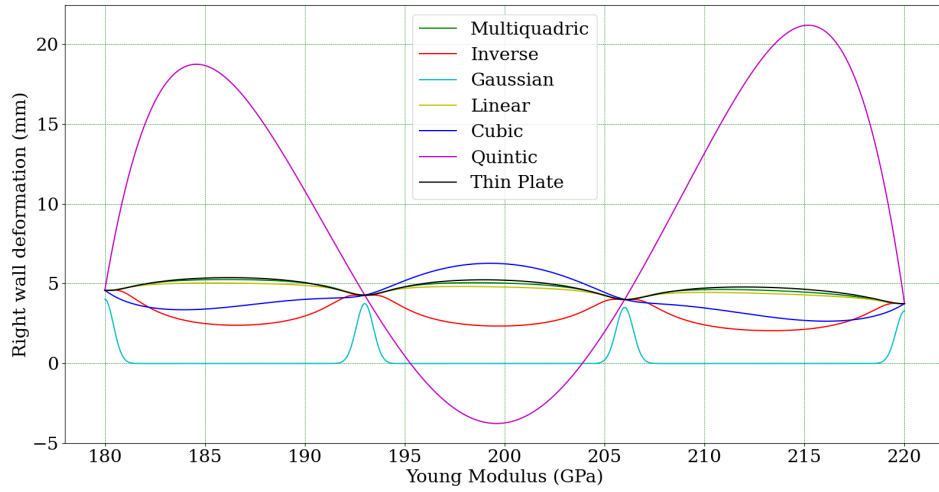


Figure 5.13 Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available function in RBF interpolant class *scipy.interpolate.Rbf*.

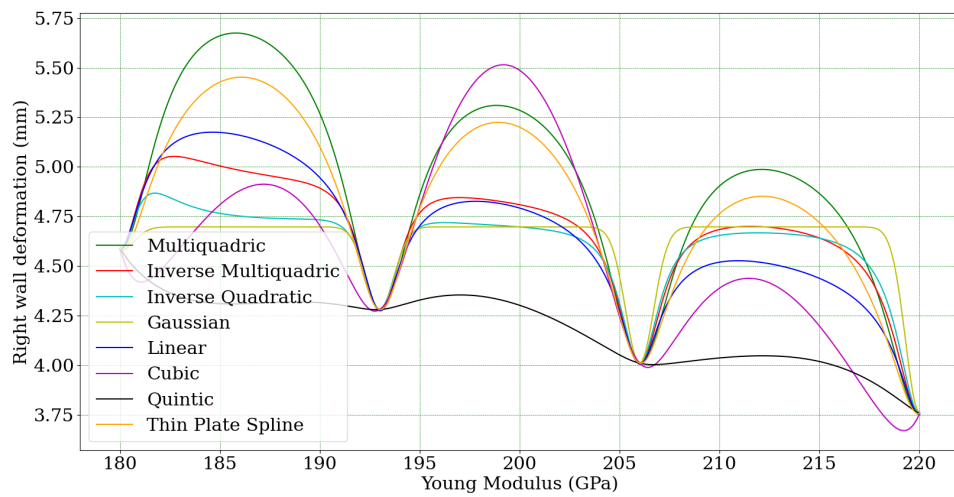


Figure 5.14 Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available kernel in RBF interpolant class *scipy.interpolate.RBF Interpolator*.

the other hand, kernels 'inverse multiquadric', 'gaussian' and 'inverse quadratic' present the worst approximations, however they are considerably better than the ones found using similar functions in *scipy.interpolate.Rbf*. Nonetheless, the most unexpected behaviour was found for the interpolation of the deformation as a function of the Young Modulus. For that case, both *scipy.interpolate.Rbf* and *scipy.interpolate.RBFInterpolator* present four irregular areas for Young Modulus values of around 180, 198, 206 and 220 GPa, which, despite not being represented here, are accentuated as the internal pressure increases. These areas coincide with the points used to generate the data set, leading to believe that the error found for these interpolation classes might be chiefly due to an incorrect interpolation when the Young Modulus is considered. This unexpected behaviour further emphasizes that neither RBF interpolant class should be used for the present study as both present an irregular and unexpected behaviour. Discovering the root cause for this behaviour is considered out of the scope of this dissertation.

As previously mentioned, and considering the unusual behaviour found for the RBF interpolant classes, a similar analysis was conducted for the MDI class. Considering both available applicable methods, 'nearest' and 'linear', the evolution of the interpolated value of the right wall's deformation, for an internal pressure of 197.5 kPa, as a function of that same wall's thickness, and as a function of the Young Modulus, is presented in figure 5.15 and in figure 5.16, respectively. Notice that the thickness of each of the remaining three walls was set to 6 mm and the Young Modulus was set to 210 GPa for the first graph while for the second graph all walls were considered to be 6 mm thick. Despite only presenting the behaviour of the interpolation functions for the right wall, a similar behaviour was found for the remaining three walls.

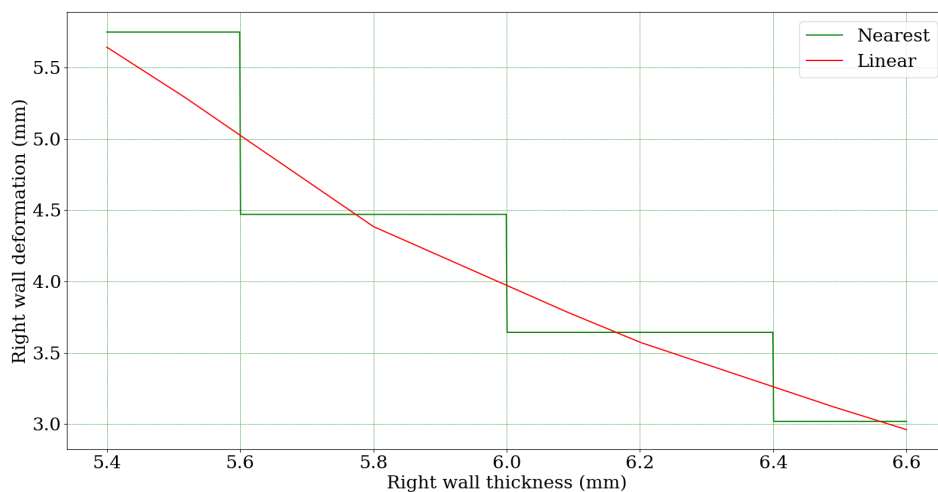


Figure 5.15 Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for an internal pressure of 197.5 kPa, using each available method in the MDI class *scipy.interpolate.griddata*.

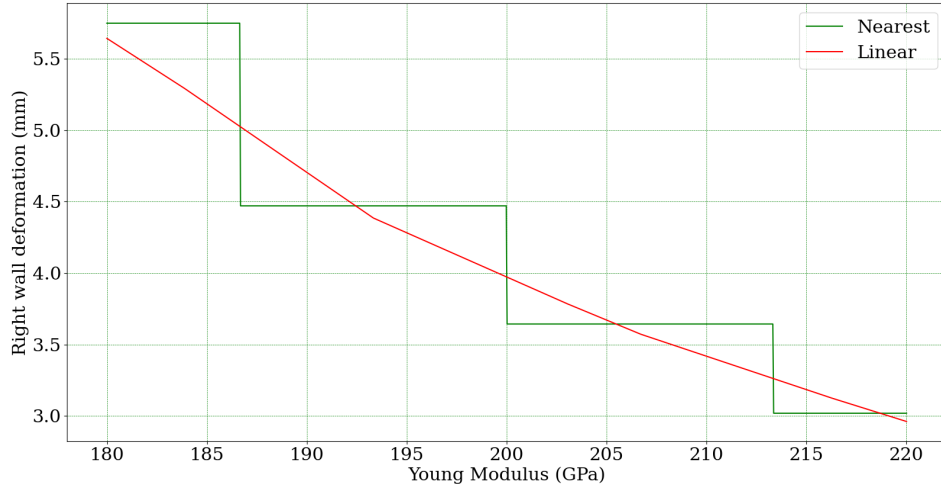


Figure 5.16 Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for an internal pressure of 197.5 kPa, using each available method in the MDI class *scipy.interpolate.griddata*.

After analysing these graphs, the contrasting behaviour of both methods ('linear' and 'nearest') stands out. While the linear method presents clear straight lines connecting the four points considered for the DOE data set ($t \in [5.4, 5.8, 6.2, 6.6]$ mm and $E \in [180, 193, 206, 220]$ GPa), the 'nearest' method presents unmistakable steps. Notice that this behaviour is not surprising for the MDI using the 'nearest' method as this method does not perform an actual interpolation but instead, for any given point, returns the value in the original data set corresponding to the point nearest the evaluated point. This accounts for the steps found in the graphs mentioned above. Notice that these steps are centered around the thickness and Young Modulus values used in the original DOE data set, thus emphasizing the behaviour described before (no interpolation is conducted, but instead the nearest value is returned).

Considering that the MDI class using the 'linear' method presented the best results, as well as a the smoothest interpolation with no evident troublesome areas, the evolution of the interpolated value for the deformation of each wall as a function of that same wall's thickness is presented for the right wall in figure 5.17, and the evolution of the interpolated value for the deformation of each wall as a function of the Young Modulus is presented for the right wall in figures 5.18. Despite not being represented, the behaviour found for each of the remaining three walls was similar. For each of the first graph the thickness of each of the remaining three walls was set to 6 mm and the Young Modulus was set to 210 GPa, while for the second graph all walls were considered to be 6 mm thick. As can be seen from these graphs, the deformation of each wall seems to be correctly interpolated for all pressure levels, both as a function of that same wall's thickness or the Young Modulus. Furthermore, no unusual behaviour is found in any of the graphs, where the expected

smooth trend is shown. Therefore, the MDI class using the 'linear' method was chosen to create the RS in the present study.

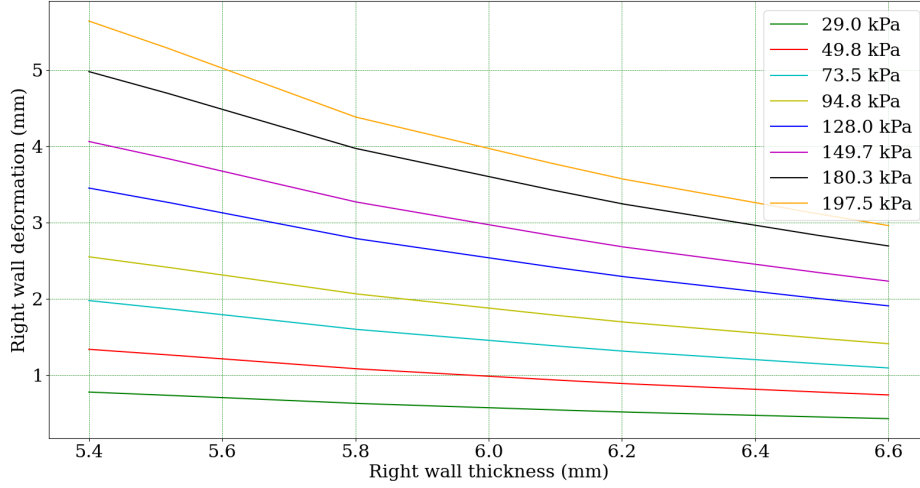


Figure 5.17 Evolution of the interpolated value for the deformation of the right wall as a function of that same wall's thickness for the MDI class, using the 'linear' method.

5.3.3 Defining the objective function

In order to conduct the SOO approaches, an objective function was created. Recall that the objective is to update the deformations in the RS (meta-model) so that they approximate the values of the deformations provided by the experimental data, allowing for the estimation of the wall thicknesses and Young Modulus. The targets for each of the deformations are presented in table 5.11, and equation 5.6 represents the objective function used for the SOO. The objective function is based on the minimum weighted sum method and aims at minimizing the residual between the deformations in the experimental data and the deformations in the model. In order to avoid placing a greater emphasis on higher pressure levels, which create greater deformations, the residual was quantified as a relative error, as is presented in equation 5.6. That being said, the objective function is a sum of the squared relative errors for each walls' deformation at each pressure level. Interestingly, note that squaring the relative error emphasizes the greater disparities between updated and target deformations, prioritizing their minimization and leading to a solution that presents an even degree of error across the various parameters under optimization.

$$ObjectiveFunction = \sum_{n=1}^{32} \left(\frac{\delta_n - \delta_{t_n}}{\delta_{t_n}} \right)^2 \quad (5.6)$$

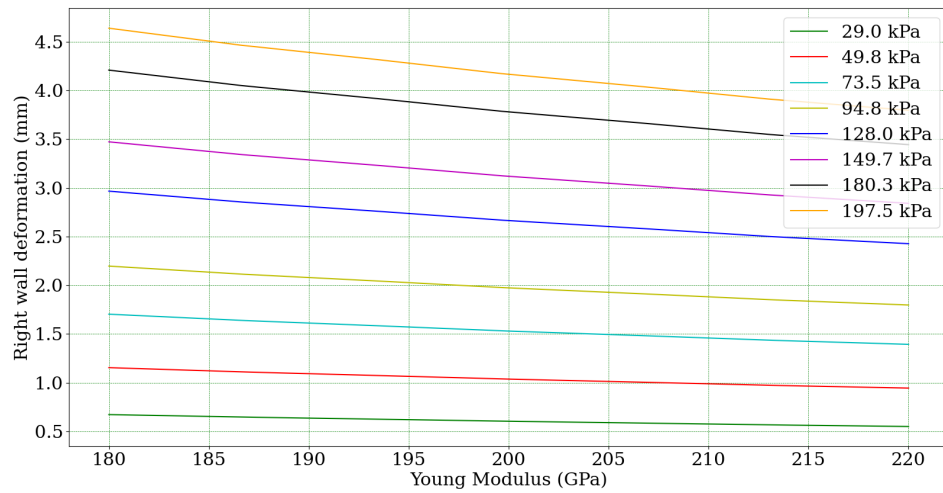


Figure 5.18 Evolution of the interpolated value for the deformation of the right wall as a function of the Young Modulus for the MDI class, using the 'linear' method.

Table 5.11 Target deformations for each wall based on the experimental data.

Model	Pressure (kPa)	Target Deformation, δ_t (mm)			
		Right Wall	Back Wall	Left Wall	Front Wall
1	29.0	0.8078	0.7114	0.7028	0.6973
2	49.8	1.4275	1.3066	1.2084	1.2482
3	73.5	2.0871	1.9574	1.7423	1.8974
4	94.8	2.6628	2.5038	2.2244	2.4474
5	128.0	3.4410	3.0740	2.8708	3.1536
6	149.7	3.9381	3.5517	3.3083	3.6701
7	180.3	4.6741	4.2084	3.9515	4.4353
8	197.5	5.0401	4.5684	4.2334	4.8228

5.3.4 Running the optimization procedures

After generating the RS, it was possible to conduct the optimization procedures. As previously mentioned, two optimization procedures were conducted: PSO and GA. The open-source algorithms used for these optimization procedures are implemented in Python and besides being extensively documented, have been recently revised and updated. Furthermore, active online forums on the subject support the verdict that these algorithms present great quality and have been thoroughly tested. Subsections 5.3.5 and 5.3.6 present detailed information regarding the implementation of PSO and GA, respectively, to the present study.

5.3.5 Implementing PSO

5.3.5.1 Introduction to PySwarms

In order to conduct the PSO, PySwarms was used. This is an extensible research toolkit in Python which provides basic customizable optimization using PSO. It is defined to solve minimization problems, however it can be used for maximization purposes by simply readjusting the objective function. It offers the possibility of conducting local and global best optimizations, but for the present study only the latter was used. The difference between these two options resides in the fact that global best optimization uses the global best position to attract all particles in the search space, while local best relies on the local best position in the neighborhood of a given particle to redirect its movement, [45].

In the global best optimization, offered by PySwarms, each particle's position is updated as follows:

$$x_i(t+1) = x_i(t) + v_i(t+1) \quad (5.7)$$

where $x_i(t)$ is the position at the current timestep, t , $x_i(t+1)$ is the updated position and $v_i(t+1)$ is the velocity, which is defined as:

$$v_i(t+1) = w \cdot v_i(t) + c_1 \cdot r_1 \cdot [pbest_i(t) - x_i(t)] + c_2 \cdot r_2 \cdot [gbest(t) - x_i(t)] \quad (5.8)$$

where $pbest_i(t)$ and $gbest(t)$ are the personal best and the global best positions at that time step, respectively, r_1 and r_2 are random numbers and c_1 and c_2 are the cognitive and social factors which govern the particle's behaviour, determining its tendency to either follow its personal best position or follow the swarm's global best position. Overall, this determines whether the swarm is exploratory or exploitative in character. In other words, the cognitive factor influences the particle's memory of its own historical experience, whereas the social factor affects the particle's information exchange and collaboration. The variables c_1 and c_2 are, therefore, learning factors. Notice that if $c_1 = 0$, there is no cognitive part but

only the social portion, and when a particle travels, it is only impacted by the historical optimum knowledge of the entire population. On the other hand, if $c_2 = 0$, the particle's movement is controlled by its own historical optimum knowledge without the interchange of information. If $c_1 = c_2 = 0$, the particle will move towards to its previous position, affected only by the inertia factor, w , [45, 46, 47].

In order to correctly call the global optimization class (`pyswarms.single.global_best.GlobalBestPSO`), some arguments must be defined, namely:

- 'n_particles': number of particles in the swarm
- 'dimensions': number of dimensions in the search space
- 'options': a dictionary containing the values for the cognitive, social and inertia factors
- bounds: upper and lower bounds for each dimension of the search space
- objective function: objective function intended to be minimized

Note that other parameters can be defined as well, allowing for greater customization of the optimization algorithm, however they were not explored in the present study.

According to [29], the population size is generally determined empirically, although numbers ranging from 20 to 50 are very frequent. In the literature, the cognitive and social factors, c_1 and c_2 , are typically set to no higher than $c_1 = c_2 = 2$. In terms of the inertia factor, w , the literature suggests that the best performance can be obtained by initially setting it to a relatively high value (for example, 0.9), which corresponds to a system in which particles move in a low viscosity medium and perform extensive exploration, and gradually lowering it to a much lower value (for example, 0.4), where the system would be more exploitative. Values greater than $w = 1$ may cause the swarm to become unstable as it will make the particle's velocity increase exponentially with each iteration, [29].

5.3.5.2 Pyswarms implementation

The PySwarms algorithm is implemented as follows, where 't_ym_opt' refers to the best candidate solution delivered by the algorithm, 'of_opt' refers to the objective function value at said point and 'of' refers to the callable objective function.

Listing 5.6 PySwarms implementation

```
import pyswarms as ps
from pyswarms.utils.plotters import (plot_cost_history,
                                     plot_contour, plot_surface)

options = {'c1': 2, 'c2': 2, 'w': 0.9}
```

```

lb = [5.4, 5.4, 5.4, 5.4, 185]
ub = [6.6, 6.6, 6.6, 6.6, 220]

constraints = lb, ub

optimizer = ps.single.GlobalBestPSO(n_particles=100,
    dimensions=5, options=options, bounds=constraints)

t_ym_opt, of_opt = optimizer.optimize(of, iters=1000)

plot_cost_history(cost_history=optimizer.cost_history)

plt.show()

```

In order to run PySwarms correctly, the callable objective function, 'of', must receive an array with a number of rows equal to the number of particles in the swarm and a number of columns equal to the dimension of the problem, and return a single column array of real values with as many rows as particles in the swarm. Therefore, for each iteration of the algorithm, the objective function is calculated for all the particles in the swarm using one single callable function. The array received by the objective function is a matrix where each line refers to a particle in the swarm and each column refers to an input variable (wall thicknesses and Young Modulus).

5.3.5.3 Conducting the optimization procedure using PySwarms

Given the fact that the problem at hands presents a great amount of variables, the swarm size was initially set to 50 particles, therefore abiding by the aforementioned guidelines. To analyse the impact of the various factors, three possible values were analysed for each of them and all combinations between said values were analysed, leading to a total of $3^3 = 27$ combinations. Afterwards, the impact of the population size was also evaluated, considering the following sizes: 25, 50, 75, 100, 200.

For the cognitive and social factors, c_1 and c_2 , values 1, 1.5 and 2 were considered. As for the inertia factor, w , values 0.4, 0.65 and 0.9 were used. The maximum number of iterations was set to 200 as this value allowed for the algorithm to converge and stabilize for the aforementioned parameters. The obtained optimal solutions are presented in table 5.12, and figures 5.19, 5.20, 5.21, 5.22 and 5.23 represent the evolution of the best candidate solution as a function of the number of iterations for each combination of factors values. The graphs from figures 5.19, 5.20, 5.21 present various combinations of c_2 and w , with $c_1 = 1$, $c_1 = 1.5$ and $c_2 = 2$, respectively. The graph from figure 5.22 presents various combinations of c_1 and c_2 , with $w = 0.9$ and the graph from figure 5.23 presents various

combinations of c_1 , c_2 and w , with an emphasis on the proportions between c_1 and c_2 . Taking into account the information presented in table 5.12, the best combination of factor was shown to be $c_1 = 1$, $c_2 = 1.5$, $w = 0.9$, so these values are used to analyse the effect of the population size. Table 5.13 presents the obtained optimal results for each swarm size (25, 50, 75, 100 or 200 particles), and figure 5.24 shows the evolution of the objective function value as a function of the number of iterations for each of these swarm sizes.

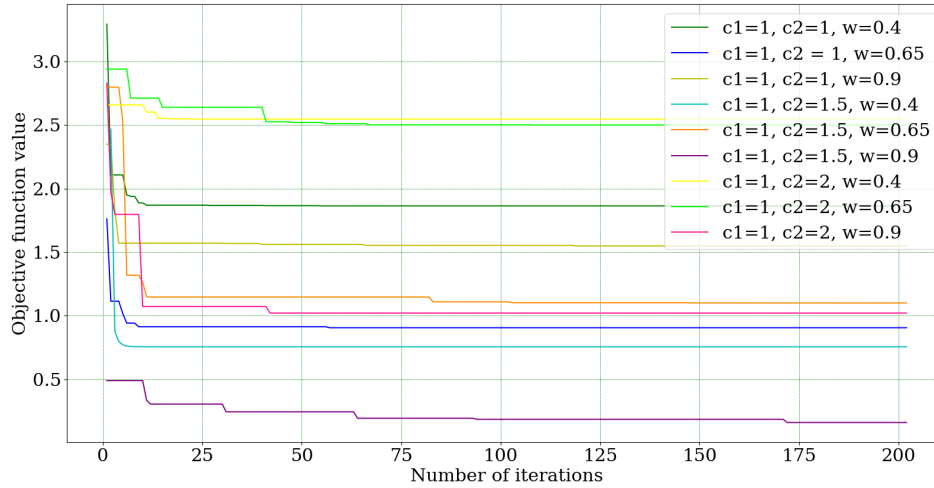


Figure 5.19 Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 1$).

5.3.5.4 Results and comments

Based on the results presented in the previous section, it is noticeable that the PSO algorithm seems to converge to different optimal values using different combinations of the three factors, c_1 , c_2 and w . This may mean that the search space is not effectively analysed for every case, which can be related to two key factors: the exploratory vs. exploitative character of the swarm and the swarms velocity. The graph from figure 5.23, which represents the evolution of the objective function value as a function of the number of iterations for various combinations of c_1 , c_2 and w with an emphasis on the proportion between c_1 and c_2 , sheds some light on these two aspects. Firstly, it is relevant to note that a given proportion between c_1 and c_2 will influence the exploratory vs. exploitative character of the algorithm, however the actual values used for these factors will influence the velocity of the swarm. This can be seen when comparing the situations where $c_1 = c_2 = 1$ and $c_1 = c_2 = 2$. For the first case, the lower values of c_1 and c_2 won't promote the acceleration of the swarm, leading to a slower exploration of the search space, and an increased likelihood of not approaching the global optima. The graph from figure 5.23 shows that when $c_1 = c_2 = 1$ a worse objective function value is reached when compared

Table 5.12 Results obtained from applying PSO using a swarm of 50 particles and limiting the number of iterations to 200.

c_1	c_2	w	t_{Front} (mm)	t_{Back} (mm)	t_{Right} (mm)	t_{Left} (mm)	Young Modulus (GPa)	Objective Function
1	1	0.4	5.8540	5.6372	6.2880	5.7454	212.7	1.863
1	1	0.65	5.8606	6.0430	5.5736	6.2276	192.5	0.905
1	1	0.9	6.2798	5.6357	5.5222	5.8763	204.0	1.547
1	1.5	0.4	5.7409	5.5712	5.4442	5.7678	198.0	0.755
1	1.5	0.65	5.9517	5.7624	5.4674	5.9941	208.8	1.100
1	1.5	0.9	5.4846	5.8537	5.6932	5.9390	187.0	0.160
1	2	0.4	6.4456	5.5947	6.0557	5.7972	209.7	2.546
1	2	0.65	6.1561	5.7742	6.4199	6.0010	198.0	2.500
1	2	0.9	5.9899	5.5260	5.4213	5.8249	207.2	1.020
1.5	1	0.4	6.5109	5.6444	5.6038	5.7402	185.7	1.867
1.5	1	0.65	5.4104	5.6642	5.4521	5.8792	205.2	0.192
1.5	1	0.9	5.5515	5.9824	5.5044	5.9597	203.7	0.536
1.5	1.5	0.4	5.6943	5.6651	6.2307	5.8525	197.4	1.183
1.5	1.5	0.65	5.4015	5.5657	6.1698	6.4371	208.9	1.708
1.5	1.5	0.9	5.4258	6.0692	5.5116	6.1008	190.0	0.422
1.5	2	0.4	5.4127	6.0407	5.6004	6.0062	200.2	0.453
1.5	2	0.65	5.7538	5.6179	5.6402	6.3065	198.7	0.981
1.5	2	0.9	5.5080	5.6807	5.6147	6.1070	192.5	0.236
2	1	0.4	6.1509	5.8418	5.9898	5.9058	186.5	1.351
2	1	0.65	5.7353	5.6140	5.6312	5.8731	196.0	0.467
2	1	0.9	5.4285	5.9345	6.2821	5.8905	192.0	0.926
2	1.5	0.4	5.4014	6.3063	5.8559	6.2978	211.8	1.811
2	1.5	0.65	5.4006	5.9723	5.7327	5.9646	188.2	0.252
2	1.5	0.9	5.4773	5.6342	5.7007	6.0024	202.9	0.342
2	2	0.4	6.0345	6.4333	5.5212	6.0514	204.0	2.157
2	2	0.65	5.6622	6.5524	5.6200	5.8827	187.3	1.300
2	2	0.9	5.4720	5.9936	5.5499	5.7721	197.9	0.400

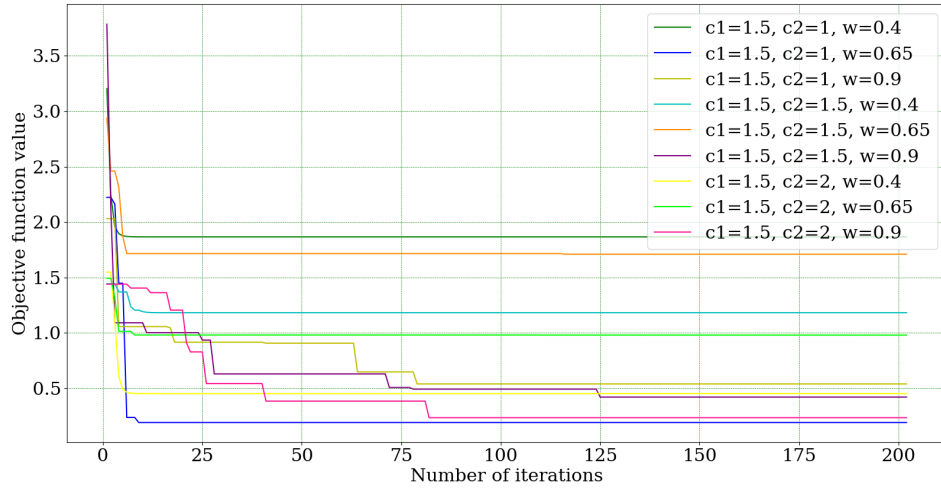


Figure 5.20 Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 1.5$).

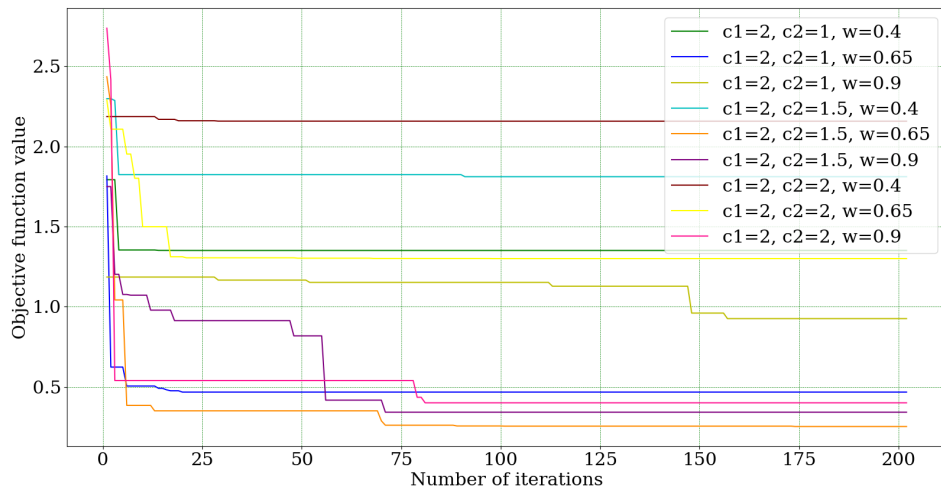


Figure 5.21 Evolution of the objective function value for various combinations of c_2 and w (no. particles = 50, max. no. iterations = 300, $c_1 = 2$).

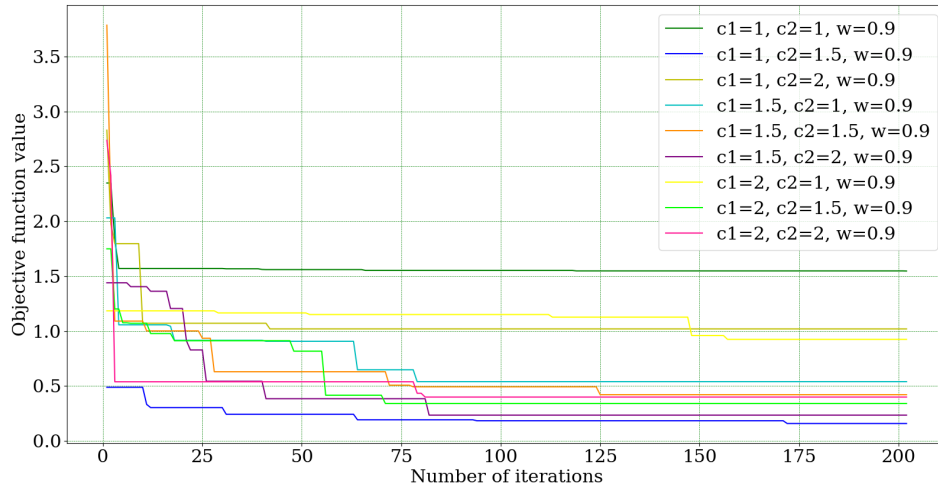


Figure 5.22 Evolution of the objective function value for various combinations of c_1 and c_2 (no. particles = 50, max. no. iterations = 300, $w = 0.9$).

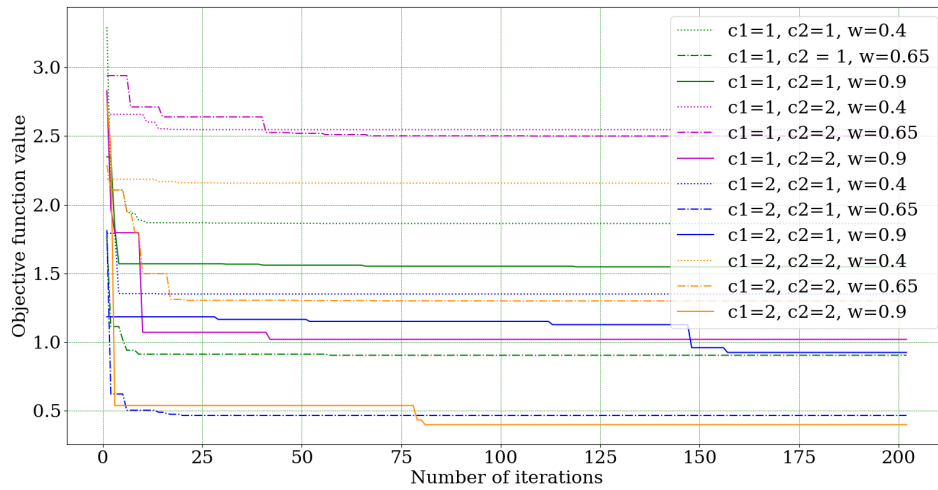


Figure 5.23 Evolution of the objective function value for various combinations of c_1 , c_2 and w , with an emphasis on the proportions between c_1 and c_2 (no. particles = 50, max. no. iterations = 300).

Table 5.13 Results obtained from applying PSO using different swarm sizes ($c_1 = 1$, $c_2 = 1.5$, $w = 0.9$).

Number of Particles	25	50	75	100	200
Maximum Iterations	200	200	200	200	200
t_{Front} (mm)	5.5027	5.4846	5.4788	5.4135	5.4287
t_{Back} (mm)	5.6289	5.8537	5.7225	5.7164	5.6566
t_{Right} (mm)	5.7011	5.6932	5.4268	5.4205	5.7226
t_{Left} (mm)	6.0209	5.9390	5.9488	5.5430	6.0495
Young Modulus (GPa)	191.2	187.0	206.9	218.0	194.2
Objective Function	0.233	0.160	0.338	0.390	0.224

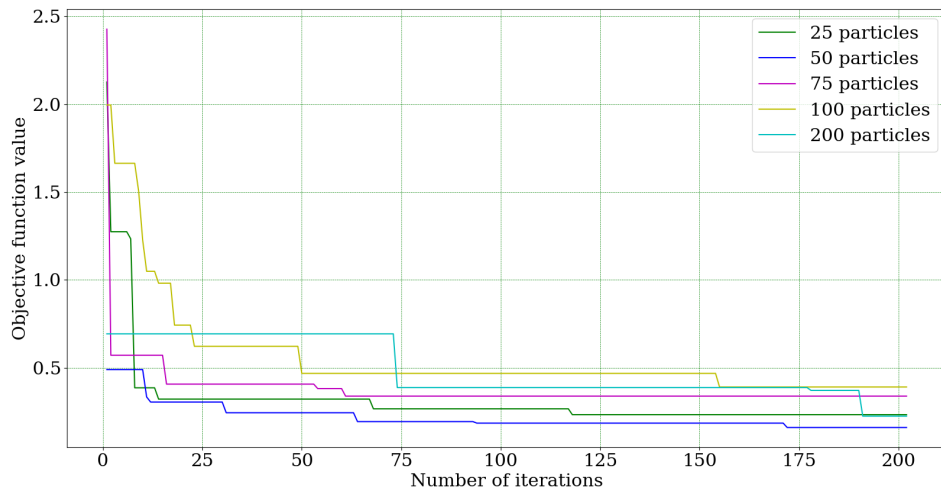


Figure 5.24 Evolution of the objective function value as a function of the population size (max. no. iterations = 200, $c_1 = 1$, $c_2 = 1.5$, $w = 0.9$)

to the case where $c_1 = c_2 = 2$ is used. This might mean that values of $c_1 = c_2 = 1$ do not promote an effective investigation of the search space due to the lower acceleration they promote. From these two situations, it is also shown that the use of a higher value of w leads to better results; $w = 0.9$ in particular seems to greatly benefit these two combinations on c_1 and c_2 , as well as a combination of $c_1 = c_2 = 1.5$, as can be seen in figure 5.20. When analysing the situation where $c_2 = 2 \times c_1 = 2$, which represents a swarm with a greater tendency to follow the global best known position, the results found for $w = 0.9$ show a worse performance and a higher optimal objective function value than the ones obtained for $c_1 = 2 \times c_2 = 2$, which represents a swarm where the particles display a greater tendency to follow their own personal best known position. This shows that a tendency to follow the particle's personal best position might be the most beneficial for the problem at hands. Nonetheless, the best objective function value was achieved using a combination of $c_2 = 1.5 \times c_1 = 1.5$, however that particular run of the algorithm started with an initial swarm with a very good global best position, as can be seen in figure 5.22, thus an approach which encourages the swarm to move towards the global best position would be more likely to reach better results.

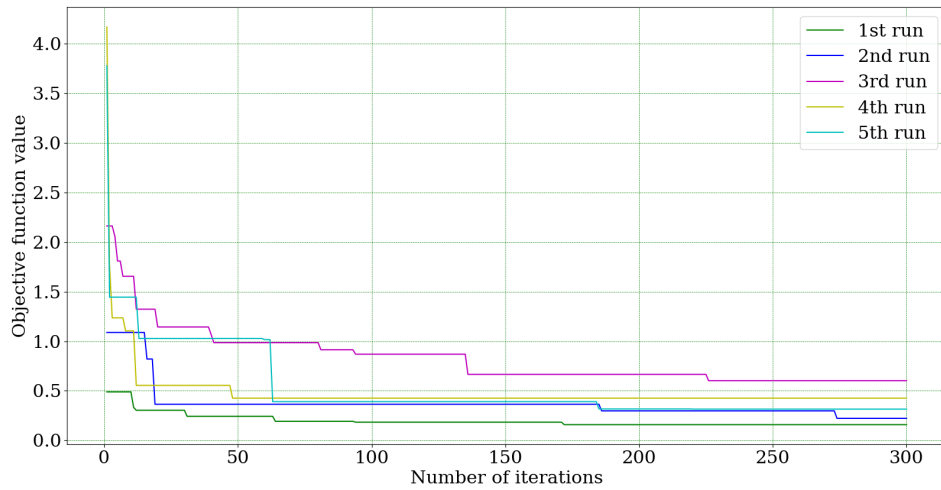
Regarding the number of particles in the swarm, a larger number of particles in the initial swarm should lead to an increased chance of having a particularly good candidate solution in the initial set. However, the graph from figure 5.24 shows that a larger swarm size does not necessarily lead to a better initial global best position, which is not surprising considering the stochastic nature of the algorithm. Furthermore, an increase in the number of design points beyond 50 does not seem to be beneficial as the obtained results are not improved by that.

Considering the results obtained, it would appear that Pyswarms is particularly unpredictable as re-running the algorithm for a given set of parameters inevitably leads to multiple results, sometimes with great disparity between them. This behaviour is displayed in table 5.14 and figure 5.25. Despite being a stochastic algorithm, the magnitude of this behaviour is not to be expected, given the fact that the RS used for the optimization appears to be smooth. Therefore, this algorithm does not seem to be the most adequate to be implemented in the present case, though further analysis of the customizable features of the algorithm could be conducted to better understand this behaviour and attempt to promote a more effective performance.

On a final note, figure 5.25 emphasizes the role that an initial swarm containing a good candidate solution plays in the quality of the obtained results given that, for the same parameter values, the runs which presented a lower objective function value for the global best candidate at the beginning of the run led to better results. This was verified for all runs except the third run, which, despite containing a better candidate solution in the initial swarm than the fourth and fifth runs, still led to worse results.

Table 5.14 Analysis of various runs of the PySwarms algorithm, using the same parameters.

Number of Particles	50	50	50	50	50
Maximum Iterations	300	300	300	300	300
c_1	1	1	1	1	1
c_2	1.5	1.5	1.5	1.5	1.5
w	0.9	0.9	0.9	0.9	0.9
t_{Front} (mm)	5.4846	5.4165	5.4595	5.5853	5.4976
t_{Back} (mm)	5.8537	5.8756	5.8513	5.5033	5.7513
t_{Right} (mm)	5.6932	5.6250	5.7760	5.4794	5.5072
t_{Left} (mm)	5.9390	6.1012	5.6468	5.8713	5.6970
Young Modulus (GPa)	187.0	189.7	212.0	208.4	206.1
Objective Function	0.160	0.223	0.602	0.427	0.316

**Figure 5.25** Analysis of various runs of the PySwarms algorithm, using the same parameters (no. particles = 50, max. no. iterations = 300, $c_1 = 1$, $c_2 = 1.5$, $w = 0.9$)

5.3.6 Implementing GA

5.3.6.1 Introduction to geneticalgorithm

For the GA optimization, the `geneticalgorithm` Python library was used. This library offers a standard elitist GA with the ability to solve continuous, combinatorial and mixed optimization problems with continuous, discrete, and mixed variables. It was created to solve minimization problems, however it can be used for maximization purposes by simply readjusting the objective function, [48].

In order to correctly call this library, various arguments must be defined, namely:

- `'function'`: objective function intended to be minimized
- `'dimensions'`: number of dimensions in the search space
- `'variable_type'`: the type of variable under analysis
- `'variable_boundaries'`: upper and lower bounds for each dimension of the search space
- `'algorithm_parameters'`: optional argument referring to additional parameters that may be customized, namely:
 - `'max_num_iteration'`: maximum number of iterations. It's the termination criterion. If this option is set to *None*, the algorithm determines the maximum number of iterations based on the dimension, boundaries, and population size. The user can input as many iterations as they like and it is strongly advised that the user determines the maximum number of iterations rather than using *None*.
 - `'population_size'`: population size. Defaults to 100.
 - `'mutation_probability'`: mutation probability. Determines the chance of each gene in each individual solution being replaced by a random value. The default is 0.1 (10%).
 - `'elit_ratio'`: elitist ratio. Determines the number of individuals preserved as elites in the population. The default value is 0.01 (1%). To implement a standard non-elitist GA, this parameter must be set to 0.
 - `'crossover_probability'`: crossover probability. Determines the chance of an existing solution passing breeding offspring. The default value is 0.5 (50%).
 - `'parents_portion'`: parents portion. Number of individuals in the population from previous generations (parents). The default value is 0.3 (30%).
 - `'crossover_typ'`: crossover type. There are three options: one-point, two-point, and uniform crossover functions. The default is uniform crossover.

- 'max_iteration_without_improv': maximum number of iterations without improvement. If the algorithm does not improve the objective function over the number of successive iterations determined by this parameter, then the algorithm stops and reports the best solution found thus far. The default value is *None*.

5.3.6.2 geneticalgorithm implementation

The geneticalgorithm is implemented as follows, where 'of' refers to the callable objective function.

Listing 5.7 geneticalgorithm implementation

```
import numpy as np
from geneticalgorithm import geneticalgorithm as ga

varbound=np.array([[5.4, 6.6], [5.4, 6.6], [5.4, 6.6],
                  [5.4, 6.6], [185, 220]])

algorithm_param = {'max_num_iteration': 3000,
                   'population_size':100,
                   'mutation_probability':0.1,
                   'elit_ratio': 0.01,
                   'crossover_probability': 0.5,
                   'parents_portion': 0.3,
                   'crossover_type':'uniform',
                   'max_iteration_without_improv':None}

model=ga(function=of,
          dimension=5,
          variable_type='real',
          variable_boundaries=varbound,
          algorithm_parameters=algorithm_param)

model.run()
```

In order to run geneticalgorithm correctly, the callable objective function, 'of', must receive a line array with a number of columns equal to the dimension of the problem, and return a scalar. Therefore, for each iteration of the algorithm, the objective function is called as many times as the number of individuals to be evaluated in that iteration. The array received by the objective function is a vector where each component refers to an input variable (wall thicknesses and Young Modulus). The value returned by the

objective function is the objective function value for each of the evaluated individuals (candidate solutions).

5.3.6.3 Conducting the optimization procedure using the geneticalgorithm

Since there is no general consensus in literature regarding recommended values or intervals for each of the GA's parameters, a similar approach to the one conducted for PySwarms was implemented. Using a population of 50 individuals and a maximum number of iterations of 200, the effect of mutation probability, crossover probability and parents portion was evaluated. Although the elite ratio is also customizable, the default was used (0.01), meaning that a single elite individual is preserved from generation to generation ($0.01 \times 50 = 0.5$ but the algorithm considers at least 1 individual). Besides this, using high values of elite ratio hinders the algorithm as it limits its capacity to inspect the search space. Regarding the mutation and crossover probabilities and the parents portion, values of 0.1, 0.3 and 0.5 were considered for each of them and all possible combinations of these parameters were analysed. The obtained optimal solutions are presented in table 5.15, and figures 5.26, 5.27, 5.26 and 5.29 represent the evolution of the best candidate solution as a function of the number of iterations for each combination of factor values. The graphs from figures 5.26, 5.27, 5.26 present various combinations of crossover probability and parents portion, with the mutation probability set to 0.1, 0.3 and 0.5, respectively. The graph from figure 5.29 presents various combinations of mutation and crossover probabilities, with the parents portion set to 0.1. Taking into account the information presented in table 5.15, the best combination of parameters was shown to be 0.3, 0.3 and 0.1, for mutation probability, crossover probability and parents portion, respectively, so these values were used to analyse the effect of the population size. Table 5.16 presents the obtained optimal results for each population size (25, 50, 75, 100 or 200 individuals), and figure 5.30 shows the evolution of the objective function value as a function of the number of iterations for each of these population sizes.

5.3.6.4 Results and comments

As can be seen from the results presented in table 5.15, the differences between the worse optimal solution and best optimal solution obtained using the geneticalgorithm for various combinations of parameter values are all less than 0.01, which, compared to what was found for PSO can be considered a marginal difference. Therefore, it would seem that the GA either does a more thorough inspection of the search space or the smoothness of the RS inhibits the manifestation of different behaviours in the algorithm as a function of the parameters' values. Either way, the obtained results were analysed as follows.

Due to the algorithms capacity to approximately reach what appears to be the global optimum for every run of the algorithm, the best individual contained in the initial population does not seem to affect the results obtained for the maximum number of iterations

Table 5.15 Results obtained from applying GA using a population of 50 individuals and limiting the number of iterations to 200.

Mutation Probability	Crossover Probability	Parents Portion	t_{Front} (mm)	t_{Back} (mm)	t_{Right} (mm)	t_{Left} (mm)	Young Modulus (GPa)	Objective Function
0.1	0.1	0.1	5.4059	5.7693	5.6196	5.9235	185.0	0.1002
0.1	0.1	0.3	5.4031	5.7299	5.5763	5.8820	192.5	0.0979
0.1	0.1	0.5	5.4069	5.7685	5.6205	5.9244	185.0	0.1000
0.1	0.3	0.1	5.4005	5.7728	5.6194	5.9199	185.0	0.1001
0.1	0.3	0.3	5.4038	5.7254	5.5703	5.8945	192.7	0.0977
0.1	0.3	0.5	5.4094	5.7273	5.5687	5.8987	192.6	0.0991
0.1	0.5	0.1	5.4020	5.7293	5.5780	5.8970	192.6	0.0972
0.1	0.5	0.3	5.4019	5.7273	5.5733	5.8968	192.5	0.0972
0.1	0.5	0.5	5.4009	5.7266	5.5709	5.8990	192.8	0.0971
0.3	0.1	0.1	5.4034	5.7661	5.6112	5.9229	185.2	0.1014
0.3	0.1	0.3	5.4028	5.7309	5.5781	5.8926	192.5	0.0975
0.3	0.1	0.5	5.4073	5.7364	5.5752	5.8898	192.4	0.0988
0.3	0.3	0.1	5.4001	5.7303	5.5732	5.9021	192.5	0.0968
0.3	0.3	0.3	5.4049	5.7356	5.5737	5.9020	192.2	0.0985
0.3	0.3	0.5	5.4010	5.7391	5.5741	5.8928	192.3	0.0974
0.3	0.5	0.1	5.4073	5.7689	5.6179	5.9229	185.1	0.1003
0.3	0.5	0.3	5.4024	5.7273	5.5651	5.8998	192.9	0.0978
0.3	0.5	0.5	5.4182	5.7348	5.5893	5.8924	192.4	0.1022
0.5	0.1	0.1	5.4021	5.7412	5.5738	5.8928	192.7	0.0977
0.5	0.1	0.3	5.4006	5.7173	5.5730	5.9031	192.1	0.0984
0.5	0.1	0.5	5.4221	5.7143	5.5497	5.9155	192.5	0.1056
0.5	0.3	0.1	5.4194	5.7688	5.6264	5.9471	185.3	0.1045
0.5	0.3	0.3	5.4435	5.7648	5.6301	5.9375	185.3	0.1068
0.5	0.3	0.5	5.4151	5.7617	5.5866	5.8791	192.5	0.1040
0.5	0.5	0.1	5.4106	5.7384	5.5623	5.8724	192.6	0.1009
0.5	0.5	0.3	5.4167	5.7517	5.5849	5.8942	192.4	0.1025
0.5	0.5	0.5	5.4220	5.7531	5.5640	5.9081	192.0	0.1049

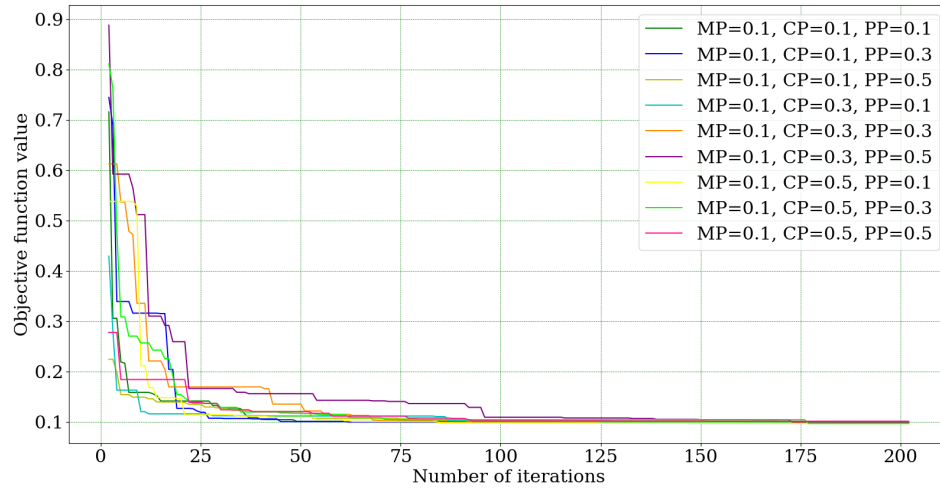


Figure 5.26 Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.1).

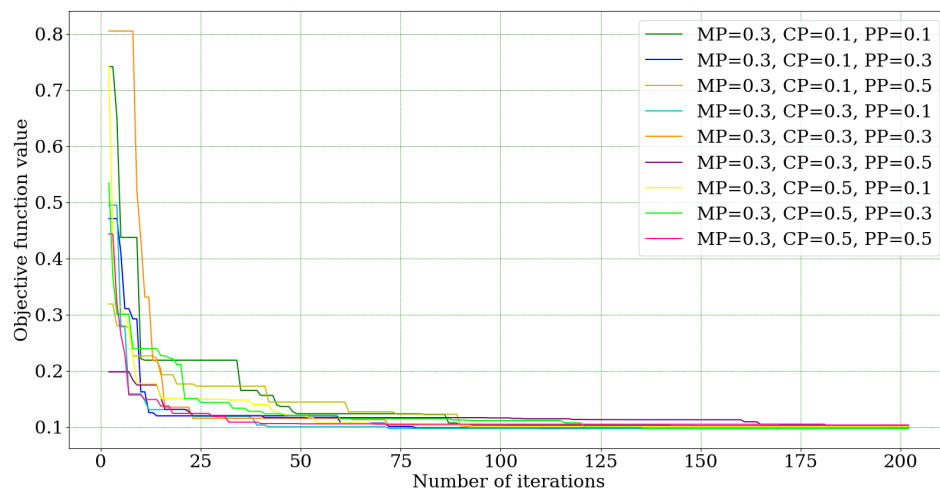


Figure 5.27 Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, Max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.3).

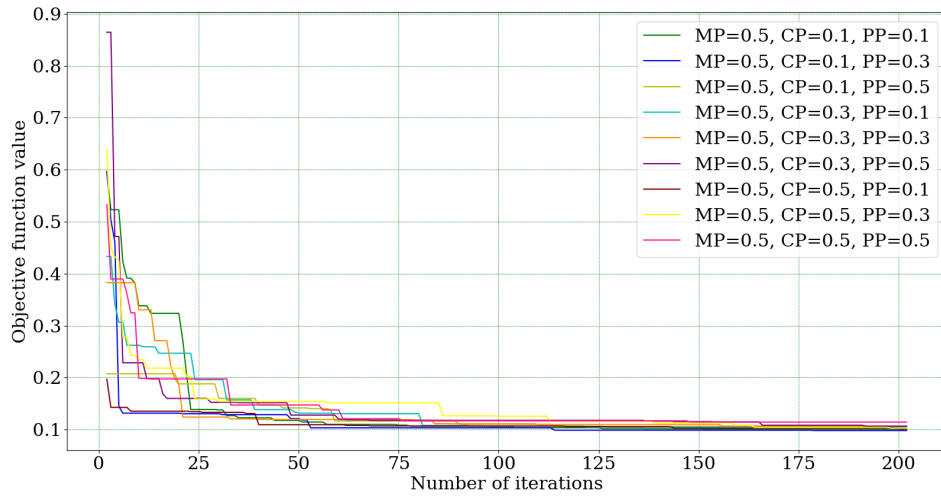


Figure 5.28 Evolution of the objective function value for various combinations of crossover probability (CP) and parents portion (PP) (population size = 50, max. no. iterations = 200, elite ratio = 0.01, mutation probability (MP) = 0.5).

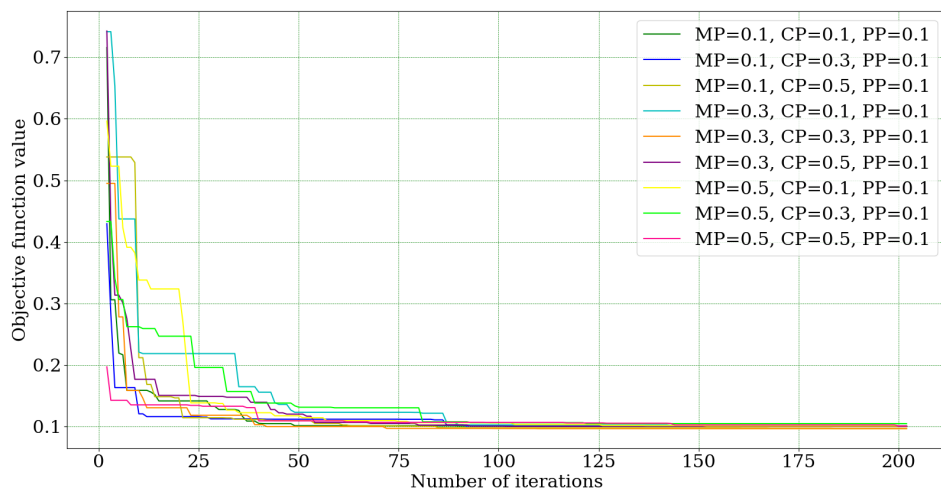


Figure 5.29 Evolution of the objective function value for various combinations of mutation probability (MP) and crossover probability (CP) (population size = 50, max. no. Iterations = 200, elite ratio = 0.01, parents portion (PP) = 0.1).

Table 5.16 Results obtained from applying GA using different population sizes (max. no. iterations = 200, mutation probability (MP) = 0.3, elite ratio = 0.01 , crossover probability (CP) = 0.3, parents portion (PP)= 0.1).

Maximum Iterations	200	200	200	200	200
Population size	25	50	75	100	200
t_{Front} (mm)	5.4022	5.4001	5.4004	5.4045	5.4017
t_{Back} (mm)	5.7246	5.7303	5.7251	5.7661	5.7286
t_{Right} (mm)	5.5718	5.5732	5.5733	5.6197	5.5727
t_{Left} (mm)	5.8985	5.9021	5.8928	5.9193	5.8941
Young Modulus (GPa)	192.6	192.5	192.6	185.1	192.6
Objective Function	0.0973	0.0968	0.0968	0.1005	0.0971

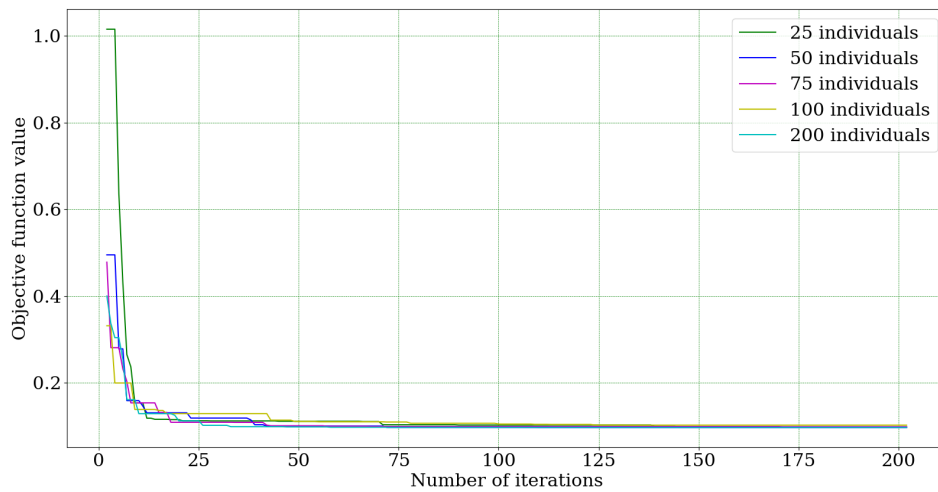


Figure 5.30 Evolution of the objective function value for various population sizes (max. no. iterations = 200, mutation probability (MP) = 0.3, elite ratio = 0.01 , crossover probability (CP) = 0.3, parents portion (PP) = 0.1).

allowed. Even for a number of iterations as low as 50, the size of the population does not seem to significantly affect the results obtained up to that point. This is shown in figure 5.30 where various population sizes are considered. Interestingly, this graph also shows the expected tendency to find better candidate solutions in larger initial samples, even if that was shown to be irrelevant in the performance of the algorithm.

On a different note, running as little as 100 iterations would seem sufficient since a higher number of iterations does not seem to benefit the obtained results significantly, as can be seen in figures 5.26, 5.27, 5.26 and 5.29.

Considering figure 5.29, one can see that a combination of a high mutation probability (0.5) with a low crossover probability and parents portion (0.1 for both) prompts the algorithm to take longer to converge; however, as the RS is smooth, intensifying the stochastic nature of the algorithm and forcing it to search random areas of the domain is not beneficial. One should note, however, that if the RS were more intricate, with a narrow plunge near the global optima, this behaviour would be extremely beneficial to the effectiveness of the algorithm.

Regarding the crossover probability, figure 5.29 shows that higher values for this parameter (0.5) may stall the algorithm by promoting a thorough local search due to a larger amount of good candidate solutions generating offspring. Lower values (0.1), on the other hand, seem to allow for global search to be conducted while still promoting local search at a lower rate, which is specially beneficial when the initial sample set does not contain any particularly good candidate solution, as can be seen for the case where the three parameters under analysis were set to 0.1.

On the other hand, and with a shift of focus towards mutation probability, for the case where mutation probability was set to 0.3 and crossover probability and parents portion were both set to 0.1, the initial set contained a similar best candidate solution to the one found when the three parameters under analysis were set to 0.1, however the approach to the optimal value is done at a slower rate, even presenting iteration intervals where no improvement can be detected, which may be a consequence of the smoothness of the RS where a random search may not be the best approach. This random search is promoted by the higher value of the mutation probability and the lower value of crossover probability, which leads to the creation of random offspring at a greater rate than offspring generated through crossover. Figure 5.29 shows that this behaviour becomes intensified as the value of the mutation probability increases. Therefore, it would seem that, for the present study, an intermediate value of mutation probability might be the most interesting as it provides a compromise between improving global search, which is beneficial when the initial sample set is not particularly good, and not hindering local search, which is necessary to accurately approach the optimal value.

Regarding the parents portion, figures 5.26, 5.27 and 5.26 show that a higher value for this parameter seems to stall the search in the early stages, as it hinders the creation of new offspring by preserving individuals from the previous generation, however it seems

beneficial later on as preserving the best individuals from the previous generation promotes local search.

That being said, given that the algorithm converged particularly fast for all analysed combinations of parameter values, leading to similarly good results for all those combinations, it is not possible to determine exactly which combination of parameter values would be the most beneficial. Notwithstanding, the best solution found occurred for mutation and crossover probabilities of 0.3 and parents portion of 0.1.

5.3.7 Results and comments

The interpolation classes analysed for creating the RS led to surprising results, further emphasizing the importance of carefully analysing the interpolation method used, as it influences the results obtained from the optimization procedures. It is relevant to note that the RBF interpolant classes both displayed an unusual behaviour when interpolating the deformation of any given wall as a function of the Young Modulus, displaying a drop in the interpolated deformation values near the Young Modulus values used to create the DOE data set. This behaviour became evident when analysing a graphic representation of the deformation of a given wall as a function of the Young Modulus, however the high values found for the maximum and average relative error using these functions already indicated that there might be a problem with the interpolation. In fact, this behaviour is alarming, particularly considering that, even when using a linear interpolation function, the interpolation between the data set points was not linear. It is relevant to note that, if such behaviour were to be found in the RS used to run the optimization procedures, a tendency to converge towards the areas of the search space closest to the DOE points would be prominent, thus negatively influencing the optimization algorithm, as it would be more likely to approach local optimal values that might not accurately represent a realistic combination of wall thicknesses, Young Modulus and wall deformations for the structure under analysis.

The MDI, on the other hand, displayed the expected behaviour for both available methods. The 'nearest' method presents clear steps for the interpolation of the deformation as a function of either the wall thickness or the Young Modulus, which was expected as this method returns the function value corresponding to the point in the DOE data set which is closest to the point under analysis. The 'linear' method, on the other hand, displayed the expected linear trend in all interpolations, as well as the lowest average relative error and second lowest average relative error, when compared to all other analysed combinations of interpolant classes and functions.

After conducting various experiments using PSO (PySwarms) and GA (geneticalgorithm), the first one's stochastic nature stands out significantly when compared to the latter. When using GA, the likelihood of reaching significantly divergent results when employing the same parameters appeared to be lower than for PSO. That being said, results obtained from PSO offer little guarantee that the used parameters are in fact responsible

for said satisfactory results. Thus, the PSO seems to be less reliable than GA for implementing in a DT, offering a lower predictability of its behaviour. However, this tendency may be reduced as a larger number of iterations is conducted. On the other hand, GA converged particularly fast and led to significantly better results than PSO across all experiments. Therefore, increasing the number of iterations used in PSO would still not make it particularly competitive for the present study.

An interesting aspect which stood out was the fact that the analysis conducted using the GA led to believe that the RS was particularly smooth, since no significance deviance was found when varying the parameter values, however the same cannot be said of PSO, where the different results obtained from the various experiments could lead to the conclusion that the RS was in fact not smooth and various local optimal values could have been found. Besides this, PSO was shown to be abundantly sensitive to the quality of the candidate solutions in the initial sample set, however a similar behaviour was not found for the GA.

Despite having conducted a general analysis of the available parameters for both PSO and GA, a more comprehensive analysis would be necessary to reach conclusions that can be extrapolated to other optimization and model updating problems. Nonetheless, the analysis conducted showed that a combination of the PSO's customizable factors which emphasizes the social character of the algorithm, making it predominantly exploratory, led to better results for the present study. For the genetic algorithm, on the other hand, a moderate mutation and crossover probability combined with a low parent portion led to the best results for the present study.

Lastly, it is relevant to note that, after conducting the aforementioned analysis, an error was found in the connections considered in the *Ansys* model used for this study. Besides the mentioned connections between the various parts of the cube, a few additional connections were created automatically between the edges of the front wall and the flanges of the right and left walls, leading to increased stiffness. This led to lower deformations found in the front wall, as is shown in figure 5.31, when compared to the deformations found in the remaining walls. Figure 5.31 represents the evolution of the deformation for the four walls as a function of the internal pressure for wall thickness of 5.4 mm for all four walls and Young modulus of 180 GPa. It is relevant to note that this error did not greatly affect the obtained results, particularly considering that the majority of the connections considered in the model are simplifications of the connections found in the real object. Since the connections between the four walls are welded, the weld bead thickness should be considered for the connections and if so, that would certainly increase the local stiffness. That being said, a more thorough analysis could be conducted with a particular emphasis on the correct representation of the connections between the various elements of the cube.

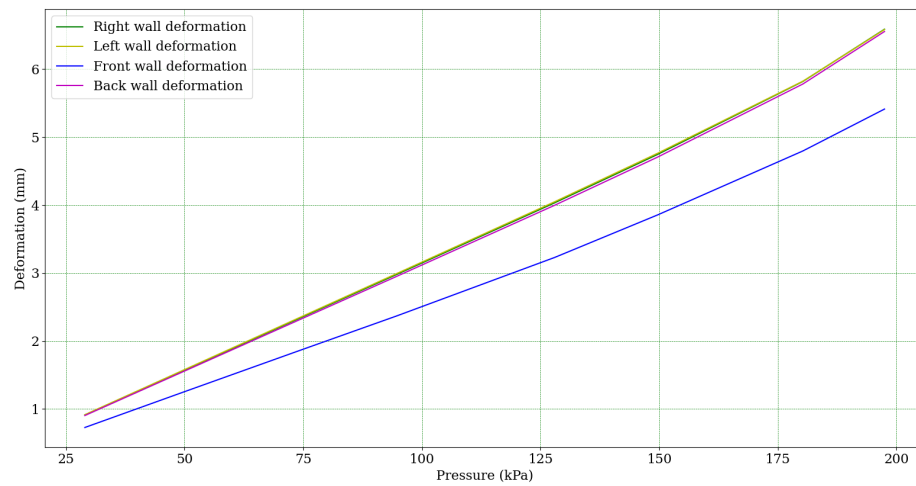


Figure 5.31 Evolution of the deformation for the four walls as a function of the internal pressure for wall thickness of 5.4 mm for all four walls and Young modulus of 180 GPa.

This page was intentionally left blank.

Chapter 6

Conclusions and future work

6.1 Main conclusions

As previously stated, the primary goal of this dissertation was to investigate, develop, and implement model updating techniques which could be used in the context of DTs. Placing a significant emphasis on the application of EAs, namely GA and PSO, various optimization approaches were implemented. The use of SOO and MOO was thoroughly analysed, as well as the use of RSO and DO. In terms of the RSO, two different approaches were evaluated, one of them completely relying on *Ansys*, and another one using a data set created in *Ansys*, but implemented in Python, where the actual RS was created and the optimization procedures conducted. Although the conclusions taken from this analysis have a limited scope of application due to the simplicity of the analysed problems, the main goal of the dissertation is considered to have been accomplished.

Regarding the comparison between RSO and DO, the first option seems to be the most interesting one for model updating, particularly in the context of DTs. First and foremost, the fact that, for RSO, the FE model must only be reevaluated to create the RS, but not to run the optimization procedure, and therefore conduct model updating, makes it particularly competitive in terms of computational cost, specially when compared with DO, which relies on reevaluating the FE model for each candidate solution analysed in the optimization process. That being said, even if the computational cost of conducting DO and RSO is similar for a single instance of model updating, where the FE model might be reevaluated a similar amount of times when creating the RS and when running the DO, the impact of reevaluating the FE model for each candidate solution considered in the DO process is magnified as the number of model updating instances increases. When implementing an optimization tool for model updating in a DT, one must bear in mind that the optimization procedure will certainly be re-conducted numerous times to allow for a precise follow-up of the structure's life-cycle. Therefore, the main focus must be on guaranteeing that the optimization procedure is light-weight and effective, thus creating an expeditious model updating tool. As previously mentioned, RSO guarantees a com-

putationally effective optimization process in the long run, meeting the DT's requirement for a light-weight procedure, however it also proved to be exceptionally effective, as was demonstrated in this dissertation. In fact, RSO presented considerably better results than DO according to various metrics which assessed the quality of the obtained solutions from diverse standpoints. Notably, the use of RSO with a SOO approach presented the best results across all analysed metrics when compared to DO using either SOO or MOO and RSO using MOO, leading to the conclusion that RSO combined with SOO may be the best option when conducting model updating. The results obtained for RSO using MOO were better than the ones obtained for DO using MOO across all analysed metrics, however DO using SOO presented surprisingly better results than RSO using MOO for a low number of design points, thus reinforcing the superior performance of SOO. Interestingly, MOO seemed to present a swift improvement with the increase in the number of design points used, however increasing the number of design points to an extent which allows MOO to present competitive results does not seem practical for model updating, particularly in DTs, as an increase in the number of design points used inevitably leads to a more computationally expensive optimization procedure. Although RSO using SOO was consistently better than all other analysed options across all experiments, one should bear in mind that the optimization procedure used for this comparison (NSGA-II, a MOGA) is stochastic and further repetitive analysis should be conducted to validate the obtained results and allow for the extrapolation of the previously mentioned conclusions.

Regarding the RSs analysed, the RS created using *Python* scripting, based on a data set with 1024 design points, presented a relative error similar to the one found for the RSs created in *Ansys* ($Max. Error_{RS\ Python} = 1.64\%$, $Max. Error_{RS125} = 4.87\%$, $Max. Error_{RS525} = 0.498\%$, $Max. Error_{RS725} = 0.301\%$, $Max. Error_{RS1050} = 0.153\%$). It is relevant to note that the RSs created in *Ansys* resorted to the sparse grid method, which introduced a greater density of refinement points in problematic areas, contributing to a more realistic meta-model. The RS created using *Python* scripting, on the other hand, was based on an approximately evenly spaced grid, with no refinement around problematic areas.

The interpolation classes analysed to be used to create the RS using *Python* scripting presented surprising results, specifically the RBF interpolant classes which created a distorted interpolation of the wall deformations as a function of the Young Modulus. Although the source of this error was not thoroughly analysed, it could be related to the different magnitudes of the various inputs (wall thicknesses and Young Modulus) and the definition of the radius used in the RBF and an attempt at solving this problem could include the normalization of the input parameters. Nonetheless, MDI led to an accurate interpolation with apparently no distortions, as it followed the expected linear behaviour when interpolating the wall deformations.

Concerning the PSO and GA implemented using *Python* scripting, the first one seemed to present a notoriously variable behaviour, reaching significantly divergent results for

different runs of the algorithm when employing the same parameters. Nonetheless, the analysis conducted showed that a combination of the PSO's customizable factors which emphasizes the social character of the algorithm, making it predominantly exploratory, led to better results for the present study. For the genetic algorithm, on the other hand, a moderate mutation and crossover probability combined with a low parent portion led to the best results for the present study. Notwithstanding, the GA displayed very little sensitivity to parameter changes, leading to similarly good results across all analysed combinations with a notoriously fast convergence rate. This might be due to the smoothness of the RS which does not contain any significant plunges, facilitating the search for the global optima. Therefore, a more complex and irregular RS would be required to effectively analyse the effects of the evaluated parameters.

6.2 Future work

Although *Ansys* includes various methods for creating RSs, only sparse grid was used in this study. It would therefore be interesting to analyse the remaining methods and evaluate their performance, particularly in terms of RS quality and running time, as these are the most pressing details concerning RS generation. Kriging, in particular, seems like an interesting approach when compared to sparse grid, as it also provides the capacity for refinement in areas of the domain which may be more challenging to model in the RS.

Regarding the interpolation algorithms analysed for Python scripting, in particular the RBF interpolant classes, a more comprehensive analysis of their behaviour could be conducted to better understand the unexpected errors found in their implementation and explore possible solutions.

On a different note, a more precise representation of the connections used in the considered experimental prototype could be conducted and a comprehensive analysis of the impact of the simplification of the connections could be executed. This would be relevant to understand the extent to which a model can be simplified with minimal loss of accuracy in the representation of the real counterpart.

Additionally, it would be interesting to explore other optimization algorithms, in particular hybrid versions, which may allow for a faster and more precise approach to the optimal point, combining the competitive features of various algorithms.

Concerning the analysis of PySwarms and geneticalgorithm for PSO and GA optimization, respectively, an in depth analysis of these could provide greater insight into their performance and allow for a better understanding of how they can be combined for a more positive outcome. An interesting analysis of these algorithms could include a repetitive analysis of the results obtained from controlled variations of the algorithm's customizable parameters. Since there is still little information in literature regarding the best approach to customize these parameters, conducting this analysis could fill an existing gap in this field of research.

Furthermore, an attempt to implement an optimization algorithm using *Python* scripting coupled to *Ansys Workbench* could be developed, allowing the automatic generation of the DOE data matrix or the exploration of other optimization algorithms, in addition to the restricted options available in *Ansys*.

Finally, it would also be interesting to analyse the local optimal search available in PySwarms and extensively compare its performance with the global optimal search. By doing so one could attempt to develop more comprehensive guidelines for the use of these two tools, particularly concerning their customizable parameters, facilitating their application.

References

- [1] J. E. Mottershead and M. I. Friswell. Model updating in structural dynamics: A survey. *Journal of Sound and Vibration*, 167(2):347–375, 1993.
- [2] N. F. Alkayem and M. Cao. Damage identification in three-dimensional structures using single-objective evolutionary algorithms and finite element model updating: evaluation and comparison. *Engineering Optimization*, 50, 2018.
- [3] N. A. Z. Abdullah, M. S. M. Sani, M. M. Rahman, and I. Zaman. A review on model updating in structural dynamics. volume 100. Institute of Physics Publishing, 12 2015.
- [4] S. Haag and R. Anderl. Digital twin–proof of concept. *Manufacturing Letters*, 15:64–66, 2018.
- [5] S. Boschert, C. Heinrich, and R. Rosen. Next generation digital twin. In *Proc. tmce*, pages 209–218. Las Palmas de Gran Canaria, Spain, 2018.
- [6] T. Marwala. *Finite element model updating using computational intelligence techniques: applications to structural dynamics*. Springer Science & Business Media, 2010.
- [7] S. Ding, H. Li, C. Su, J. Yu, and F. Jin. Evolutionary artificial neural networks: a review. *Artificial Intelligence Review*, 39(3):251–260, 2013.
- [8] M. H. Kalos and P. A. Whitlock. *Monte Carlo Methods: Second Edition*. John Wiley & Sons, 2009.
- [9] J. M. Hammersley, D. C. Handscomb, and G. Weiss. Monte carlo methods. *Physics Today*, 18, 1965.
- [10] E. Rodriguez. Bayesian analysis. <https://www.britannica.com/science/Bayesian-analysis/additional-infohistory>
Consulted on 05/03/2021.
- [11] T. Marwala and S. Sibisi. Finite element model updating using bayesian framework and modal properties. *Journal of Aircraft*, 42(1):275–278, 2005.
- [12] A. Abbas, L. Zhang, and S. U. Khan. A survey on context-aware recommender systems based on computational intelligence techniques. *Computing*, 97(7):667–690, 2015.
- [13] E. Simoen, G. De Roeck, and G. Lombaert. Dealing with uncertainty in model updating for damage assessment: A review. *Mechanical Systems and Signal Processing*, 56:123–149, 2015.

- [14] M. Beer. Uncertain structural design based on nonlinear fuzzy analysis. *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik: Applied Mathematics and Mechanics*, 84(10-11):740–753, 2004.
- [15] M. Beer and M. Liebscher. Designing robust structures—a nonlinear simulation based approach. *Computers & Structures*, 86(10):1102–1122, 2008.
- [16] I. Boulkaibet, T. Marwala, M. I. Friswell, H. H. Khodaparast, and S. Adhikari. Fuzzy finite element model updating using metaheuristic optimization algorithms. In *Special Topics in Structural Dynamics, Volume 6*, pages 91–101. Springer, 2017.
- [17] L. M. Ochoa-Estopier, M. Jobson, L. Chen, C. A. Rodriguez-Forero, and R. Smith. Optimization of heat-integrated crude oil distillation systems. part ii: Heat exchanger network retrofit model. *Industrial & Engineering Chemistry Research*, 54(18):5001–5017, 2015.
- [18] A. M. Matos, J. M. Guedes, K. P. Jayachandran, and H. C. Rodrigues. Computational model for power optimization of piezoelectric vibration energy harvesters with material homogenization. *Computers & Structures*, 192:144–156, 2017.
- [19] M. A. Corne and D. Lones. *Evolutionary Algorithms*. Springer International Publishing, 2018.
- [20] M. T. Özsu. *Encyclopedia of Information Systems*. 2003.
- [21] A. Petrowski and S. B. Hamida. *Metaheuristics*. Springer, 2016.
- [22] I. Boulkaibet, L. Mthembu, T. Marwala, M. I. Friswell, and S. Adhikari. Finite element model updating using an evolutionary markov chain monte carlo algorithm.
- [23] M. M. Rai. Single-and multiple-objective optimization with differential evolution and neural networks. *VKI lecture series: introduction to optimization and multidisciplinary design*, 58, 2006.
- [24] P. A. Vikhar. Evolutionary algorithms: A critical review and its future prospects. In *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pages 261–265, 2016.
- [25] X. S. Yang, S. F. Chien, and T. O. Ting. *Bio-Inspired Computation in Telecommunications*. Morgan Kaufmann, 2015.
- [26] T. Bäck, D. B. Fogel, and Z. Michalewicz. *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018.
- [27] J. H. Holland. Genetic algorithms. *Scientific american*, 267(1):66–73, 1992.
- [28] P. Siarry, editor. *Metaheuristics*, volume 1. Springer International Publishing, Cham, 1 edition, 2016.
- [29] R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization: An overview. *Swarm Intelligence*, 1(1):33–57, 2007.

- [30] Peter J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*. Springer Berlin Heidelberg, 1998.
- [31] J. Robinson, S. Sinton, and Y. Rahmat-Samii. Particle swarm, genetic algorithm, and their hybrids: optimization of a profiled corrugated horn antenna. In *IEEE Antennas and Propagation Society International Symposium (IEEE Cat. No. 02CH37313)*, volume 1, pages 314–317. IEEE, 2002.
- [32] P. Punia and M. Kaur. Various genetic approaches for solving single and multi-objective optimization problems: a review. *International Journal*, 3(7), 2013.
- [33] S. Bandyopadhyay and S. Saha. *Unsupervised classification: similarity measures, classical and metaheuristic approaches, and applications*. Springer Science & Business Media, 2012.
- [34] S. V. Kulkarni and S. A. Khaparde. *Transformer engineering: design, technology, and diagnostics*. CRC press, 2017.
- [35] S. V. Kulkarni and S. A. Khaparde. *Transformer engineering: design and practice*, volume 25. CRC press, 2004.
- [36] M. Banović and J. Sanchez. Basics of power transformers. *Transformers Magazine*, 1(1):12–15, 2014.
- [37] Atlantic City Electric. Infrastructure 101. <https://www.atlanticcityelectric.com/SafetyCommunity/Education/Pages/Infrastructure101.aspx> Consulted on 19/04/2021.
- [38] Abaqus. About shell elements. <https://abaqus-docs.mit.edu/2017/English/SIMACAEELMRefMap/simaelm-c-shelloverview.html> Consulted on 14/05/2021.
- [39] Ansys Inc. DesignXplorer User’s Guide, 2020.
- [40] W. Du Toit. *Radial basis function interpolation*. PhD thesis, Stellenbosch: Stellenbosch University, 2008.
- [41] G. E. Fasshauer. *Meshfree approximation methods with MATLAB*, volume 6. World Scientific, 2007.
- [42] G. Wahba. *Spline models for observational data*. SIAM, 1990.
- [43] T. Hines. Rbf interpolation. <https://rbf.readthedocs.io/en/latest/interpolate.html#rbf.interpolate.KNearestRBFInterpolator> Consulted on 04/06/2021.
- [44] The SciPy Community. Scipy.interpolate. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.RBFInterpolator.html#scipy.interpolate.RBFInterpolator> Consulted on 04/06/2021.
- [45] L. J. V. Miranda. Pyswarms documentation. <https://pyswarms.readthedocs.io/en/latest/> Consulted on 05/06/2021.

- [46] C. Wang and W. Song. A modified particle swarm optimization algorithm based on velocity updating mechanism. *Ain Shams Engineering Journal*, 10(4):847–866, 2019.
- [47] J. Barrera, O. Álvarez-Bajo, J. Flores, and C. A. Coello. Limiting the velocity in the particle swarm optimization algorithm. *Computación y Sistemas*, 20(4):635–645, 2016.
- [48] R. Solgi. geneticalgorithm 1.0.2. <https://pypi.org/project/geneticalgorithm/>
Consulted on 05/06/2021.

Appendices

Appendix A

Optimization results for the RS with 125 refinement points

This appendix presents the results obtained from the SOO and MOO procedures conducted in *ANSYS* on the RS with 125 refinement points.

Table A.1 Results from the SOO experiments using the RS with 125 design points.

Single Objective Optimization			
Parameter	Unit	Value	Error (%)
$t_{TopWall}$	mm	6.2155	-0.25
$t_{BottomWall}$	mm	5.9145	-0.25
$t_{FrontWall}$	mm	5.7082	-0.14
$t_{BackWall}$	mm	6.1157	-0.26
$t_{RightWall}$	mm	5.8139	-0.24
$t_{LeftWall}$	mm	6.3146	-0.23
CAE	%	-	1.37
Young Modulus	GPa	209.4	0.22
CAEYM	%	-	1.59
Sum Sq. Diff.	mm ²	1.02×10^{-4}	-

Table A.2 Results regarding the wall thicknesses and Young Modulus obtained from the MOO experiment using the RS with 125 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.3789	-2.89	6.2731	-1.18	6.4136	-3.45
$t_{BottomWall}$	mm	6.0565	-2.65	5.9658	-1.12	6.0800	-3.05
$t_{FrontWall}$	mm	5.8587	-2.78	5.7727	-1.28	5.8563	-2.74
$t_{BackWall}$	mm	6.2713	-2.81	6.1749	-1.23	6.2895	-3.11
$t_{RightWall}$	mm	5.9554	-2.68	5.8627	-1.08	5.9788	-3.08
$t_{LeftWall}$	mm	6.5249	-3.57	6.2403	0.95	6.5413	-3.83
CAE	%	-	17.38	-	6.83	-	19.26
Young Modulus	GPa	194.6	7.31	203.7	2.96	191.7	8.67
CAEYM	%	-	24.68	-	9.79	-	27.93 4

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2701	-1.13	6.2707	-1.14	6.3973	-3.18
$t_{BottomWall}$	mm	5.9645	-1.09	5.9640	-1.09	6.0856	-3.15
$t_{FrontWall}$	mm	5.7737	-1.29	5.7726	-1.27	5.8823	-3.20
$t_{BackWall}$	mm	6.1764	-1.25	6.1901	-1.48	6.2855	-3.04
$t_{RightWall}$	mm	5.8608	-1.05	5.8604	-1.04	5.9608	-2.77
$t_{LeftWall}$	mm	6.2407	0.94	6.4980	-3.14	6.5332	-3.70
CAE	%	-	6.76	-	9.16	-	19.04
Young Modulus	GPa	203.7	2.96	203.5	3.03	193.1	8.01
CAEYM	%	-	9.72	-	12.19	-	27.05

Table A.3 Sum of the squared differences for deformations in the MOO experiment using the RS with 125 design points.

Multiple Objective Optimization								
Candidate Solution		1		2		3		
	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Wall								
Top	1.6933	3.76×10^{-8}	1.6926	8.25×10^{-7}	1.6932	7.77×10^{-8}		
Bottom	2.1895	5.88×10^{-7}	2.1878	7.49×10^{-7}	2.1977	8.10×10^{-5}		
Front	1.7040	4.17×10^{-5}	1.6923	2.65×10^{-5}	1.7321	1.20×10^{-3}		
Back	1.8389	4.96×10^{-5}	1.8424	1.26×10^{-5}	1.8486	7.28×10^{-6}		
Right	2.1678	5.83×10^{-7}	2.1730	3.66×10^{-5}	2.1720	2.52×10^{-5}		
Left	1.6650	1.82×10^{-3}	1.8151	1.15×10^{-2}	1.6759	1.01×10^{-3}		
Sum Sq. Diff. (mm ²)	-	1.91×10^{-3}	-	1.16×10^{-2}	-	2.32×10^{-3}		
Candidate Solution		4		5		6		
	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Wall								
Top	1.6949	2.03×10^{-6}	1.6948	1.80×10^{-6}	1.6931	1.83×10^{-7}		
Bottom	2.1892	2.71×10^{-7}	2.1902	2.34×10^{-6}	2.1766	1.47×10^{-4}		
Front	1.6913	3.81×10^{-5}	1.6963	1.43×10^{-6}	1.6969	3.54×10^{-7}		
Back	1.8409	2.47×10^{-5}	1.8300	2.54×10^{-4}	1.8395	4.14×10^{-5}		
Right	2.1750	6.45×10^{-5}	2.1772	1.04×10^{-4}	2.1778	1.16×10^{-4}		
Left	1.8146	1.14×10^{-2}	1.6122	9.11×10^{-3}	1.6706	1.37×10^{-3}		
Sum Sq. Diff. (mm ²)	-	1.16×10^{-2}	-	9.48×10^{-3}	-	1.68×10^{-3}		

This page was intentionally left blank.

Appendix B

Optimization results for the RS with 725 refinement points

This appendix presents the results obtained from the SOO and MOO procedures conducted in *ANSYS* on the RS with 725 refinement points.

Table B.1 Results from the SOO experiments using the RS with 725 design points.

Single Objective Optimization			
Parameter	Unit	Value	Error (%)
$t_{TopWall}$	mm	6.2817	-1.32
$t_{BottomWall}$	mm	5.9710	-1.20
$t_{FrontWall}$	mm	5.7639	-1.12
$t_{BackWall}$	mm	6.1740	-1.21
$t_{RightWall}$	mm	5.8690	-1.19
$t_{LeftWall}$	mm	6.3691	-1.10
CAE	%	-	7.14
Young Modulus	GPa	2031.1	3.23
CAEYM	%	-	10.38
Sum Sq. Diff.	mm ²	7.40×10^{-5}	-

Table B.2 Results regarding the wall thicknesses and Young Modulus obtained from the MOO experiment using the RS with 725 design points.

Multiple Objective Optimization							
Candidate Solution		1		2		3	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.1228	1.25	6.1357	1.04	6.2485	-0.78
$t_{BottomWall}$	mm	5.8183	1.38	5.8569	0.73	5.9906	-1.53
$t_{FrontWall}$	mm	5.6136	1.52	5.6654	0.61	5.7851	-1.49
$t_{BackWall}$	mm	6.0192	1.32	6.0625	0.62	6.1905	-1.48
$t_{RightWall}$	mm	5.7154	1.46	5.7621	0.65	5.8842	-1.45
$t_{LeftWall}$	mm	6.2330	1.06	6.2624	0.60	6.4474	-2.34
CAE	%	-	7.99	-	4.24	-	9.09
Young Modulus	GPa	219.0	-4.34	214.7	-2.29	200.6	4.42
CAEYM	%	-	12.33	-	6.52	-	13.51

Candidate Solution		4		5		6	
Parameter	Unit	Value	Error (%)	Value	Error (%)	Value	Error (%)
$t_{TopWall}$	mm	6.2485	-0.78	6.1081	1.48	6.1459	0.87
$t_{BottomWall}$	mm	5.9906	-1.53	5.8652	0.59	5.8638	0.61
$t_{FrontWall}$	mm	5.7851	-1.49	5.6609	0.69	5.6483	0.91
$t_{BackWall}$	mm	6.1905	-1.48	6.0688	0.51	6.0620	0.62
$t_{RightWall}$	mm	5.8842	-1.45	5.7514	0.84	5.7479	0.90
$t_{LeftWall}$	mm	6.4474	-2.34	6.2829	0.27	6.2167	1.32
CAE	%	-	9.09	-	4.38	-	5.24
Young Modulus	GPa	200.6	4.42	214.1	-1.99	214.3	-2.08
CAEYM	%	-	13.51	-	6.37	-	7.32

Table B.3 Sum of the squared differences for deformations in the MOO experiment using the RS with 725 design points.

Multiple Objective Optimization						
Candidate Solution		1		2		3
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.6821	1.30×10^{-4}	1.7059	1.54×10^{-4}	1.7346	1.69×10^{-3}
Bottom	2.1893	3.70×10^{-7}	2.1896	7.56×10^{-7}	2.1891	1.81×10^{-7}
Front	1.7005	9.29×10^{-6}	1.6899	5.79×10^{-5}	1.7046	5.00×10^{-5}
Back	1.8423	1.31×10^{-5}	1.8399	3.60×10^{-5}	1.8499	1.61×10^{-5}
Right	2.1719	2.45×10^{-5}	2.1627	1.85×10^{-5}	2.1735	4.18×10^{-5}
Left	1.6911	2.76×10^{-4}	1.7015	3.88×10^{-5}	1.6688	1.52×10^{-3}
Sum Sq. Diff. (mm ²)	-	4.53×10^{-4}	-	3.06×10^{-4}	-	3.32×10^{-3}

Multiple Objective Optimization						
Candidate Solution		4		5		6
Wall	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)	δ_{max} (mm)	Sq. Diff. (mm ²)
Top	1.7346	1.69×10^{-3}	1.7322	1.50×10^{-3}	1.7018	6.83×10^{-5}
Bottom	2.1891	1.81×10^{-7}	2.1870	3.02×10^{-6}	2.1871	2.68×10^{-6}
Front	1.7046	5.00×10^{-5}	1.6987	1.36×10^{-6}	1.7081	1.12×10^{-4}
Back	1.8499	1.61×10^{-5}	1.8391	4.60×10^{-5}	1.8440	3.68×10^{-6}
Right	2.1735	4.18×10^{-5}	2.1809	1.93×10^{-4}	2.1832	2.64×10^{-4}
Left	1.6688	$.52 \times 10^{-3}$	1.6897	3.25×10^{-4}	1.7421	1.19×10^{-3}
Sum Sq. Diff. (mm ²)	-	3.32×10^{-3}	-	2.07×10^{-3}	-	1.64×10^{-3}