DEVELOPMENT OF AN ADAPTIVE MULTIGRID SCHEME FOR COMPUTATIONAL FLUID DYNAMICS

PhD Dissertation

Mustafa Serdar TEKÇE

Eskişehir 2024

DEVELOPMENT OF AN ADAPTIVE MULTIGRID SCHEME FOR COMPUTATIONAL FLUID DYNAMICS

Mustafa Serdar TEKÇE

PhD Dissertation

Department of Airframe and Powerplant Maintenance

Supervisor: Prof. Dr. Kürşad Melih GÜLEREN

Eskişehir

Eskişehir Technical University

Institute of Graduate Programs

January 2024

FINAL APPROVAL FOR THESIS

This thesis titled DEVELOPMENT OF AN ADAPTIVE MULTIGRID SCHEME FOR COMPUTATIONAL FLUID DYNAMICS has been prepared and submitted by Mustafa Serdar TEKÇE in partial fulfillment of the requirements in “Eskişehir Technical University Directive on Graduate Education and Examination” for the Degree of PhD in Airframe and Powerplant Maintenance Department has been examined and approved on 12/01/2024.

Committee Members		**Title, Name and Surname**	**Signature**
Member	:	Prof. Dr. Kürşad Melih GÜLEREN	
Member	:	Prof. Dr. Mehmet Şerif KAVSAOĞLU	
Member	:	Assoc. Prof. Dr. Bahadır DOĞAN	
Member	:	Asst. Prof. Dr. Uğur ÖZDEMİR	
Member	:	Asst. Prof. Dr. ZAFER ÖZNALBANT	

Prof. Dr. Semra KURAMA
Director of the Institute of Graduate Programs

12/01/2024

SUPERVISOR APPROVAL

PhD student Mustafa Serdar TEKÇE, whom I supervise, has completed this thesis titled DEVELOPMENT OF AN ADAPTIVE MULTIGRID SCHEME FOR COMPUTATIONAL FLUID DYNAMICS. According to my inspections, the work is scientifically and ethically appropriate for the student to the thesis defense exam.

Supervisor
Prof. Dr. Kürşad Melih GÜLEREN

ABSTRACT

DEVELOPMENT OF AN ADAPTIVE MULTIGRID SCHEME FOR COMPUTATIONAL FLUID DYNAMICS

Mustafa Serdar TEKÇE

Department of Airframe and Powerplant Maintenance

Eskişehir Technical University, Institute of Graduate Programs, January 2024

Supervisor: Prof. Dr. Kürşad Melih GÜLEREN

Lack of variety of adaptive multigrid schemes were spotted and motivated the thesis. Main objective was to develop a new adaptive multigrid scheme while keeping robustness and performance as possible. In order to justify a fair numerical experimentation setup, authentic code generation effort was given that implements basic finite volume methods, intergrid operations and iterative solvers. A fundamental reference case was selected in order to get proper validation which had analytical Laplace solution. Fixed and adaptive multigrid schemes and algorithms were also implemented. All methods were presented with both theory and practice in a harmonized manner. All procedures and algorithms were validated regarding the reference case. Reference work unit was defined and was suggested as a new non-dimensional unit of computational cost measurement. Results were compiled and presented in an increasingly complex order. Explorations varying grid resolutions, boundary conditions and fixed multigrid schemes revealed key factors regarding reference case and multigrid methods. Exploration of adaptive multigrid schemes revealed key aspects regarding cost and performance while enabling selection of best parameter sets for the reference case. Explorations established the basis of comparison metrics. Innovative aspect of the thesis was carried out by developing a new and an alternative adaptive multigrid scheme. Comparative performances of multigrid schemes revealed that newly developed adaptive multigrid scheme is robust and is performing as well as the other ones. The new adaptive multigrid scheme was called PID driven and was shown to have 18% lower cost compared to flexible scheme while preserving performance with same effectivity.

Keywords: Multigrid, CFD, Algorithm, Scheme, Performance.

ÖZET

HESAPLAMALI AKIŞKANLAR DINAMİĞİ İÇİN BİR ADAPTİF ÇOKLU AĞ ŞEMASI GELİŞTİRİLMESİ

Mustafa Serdar TEKÇE

Uçak Gövde Motor Bakım Anabilim Dalı

Eskişehir Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Ocak 2024

Danışman: Prof. Dr. Kürşad Melih GÜLEREN

Adaptif çoklu ağ şemalarındaki çeşitlilik eksikliği görülerek tez çalışmalarına başlanıldı. Mümkün olduğu ölçüde gürbüzlüğü ve başarımı sağlayarak yeni bir adaptif çoklu ağ geliştirilmesi ana amaç olarak belirlendi. Makul bir sayısal deney düzeneği kurmak adına, temel seviyede sonlu hacimler yöntemi, ağlararası işlemler ve yinelemeli çözücüler içeren özgün kod geliştirme çabası verildi. Analitik Laplace çözümü bulunan temel bir referans vaka seçilerek, düzgün bir doğrulama hedeflendi. Sabit ve adaptif çoklu ağ şemaları ve algoritmaları uygulandı. Tüm yöntemler, teorileri ve uygulamaları ile birlikte harmanlanarak verildi. Tüm işlem ve yöntemler referans vaka özelinde doğrulandı. Referans iş birimi tanımlanarak hesaplama maliyeti için yeni bir ölçüt olarak önerildi. Sonuçlar derlendi ve artan karmaşıklık sırasıyla sunuldu. Farklı çözüm ağı çözünürlükleri, sınır koşulları ve sabit şemalar etrafında yapılan keşifler, referans vaka ve çoklu ağ yöntemleri ile alakalı önemli hususları ortaya çıkardı. Adaptif çoklu ağ şemaları etrafında yapılan keşifler, maliyet ve başarım ile alakalı önemli hususları ortaya çıkarırken, aynı zamanda en iyi örneklemlerin seçimini sağladı. Tüm keşifler karşılaştırma niceliklerinin temelini oluşturdu. Tezin yenilikçi tarafı, yeni ve alternatif bir adaptif çoklu ağ şeması geliştirilmesiyle birlikte yerine getirildi. Çoklu ağ şemalarının karşılaştırmalı performansları, yeni geliştirilen adaptif çoklu ağ şemasının, daha iyi değilse bile en az diğerleri kadar gürbüz ve başarılı olduğunu gösterdi. Yeni adaptif çoklu ağ şemasına PID sürüşlü ismi verildi ve aynı etkinlikte bir başarımda olduğu halde, esnek şemaya göre %18 daha az maliyete sahip olduğu gösterildi.

Anahtar Sözcükler: Çoklu Ağ, HAD, Algoritma, Şema, Başarım.

ACKNOWLEDGEMENTS

Research journey of this dissertation has been marked by persistence, from the initial conceptualization to the final realization. Health issues, complications at workplace, responsibility of the family and schedule concerns were some of the challenges that have been coped with. The last few years, though the most difficult of my life yet, have been an amazing stage for growth.

Every person and institution I have encountered has made their respective contributions. I thank to anyone that has affected the journey positively in any form, at any scale. Acknowledging the impossibility of mentioning everyone individually, I sincerely thank to all who enriched the experience. Interactions with some of my colleagues also deserve gratitude. I appreciate the exchange of ideas, shared passion of the field and collaborative attitude.

Prof. Dr. Kürşad Melih GÜLEREN transcended the role of a supervisor, providing mentorship, insightful feedback, and guidance that mitigated academic challenges with patience and expertise.

Thanks to my mother, Hacer TANRIVERDİ, whose endless devotion paved the way for every achievement. I simply can’t thank her enough in any imaginable way.

Thanks to my spouse, Çiğdem YILDIRIM TEKÇE, who provided support beyond measure with understanding and a nurturing foundation, sharing the weight of the world. I feel more than lucky to have her in my life.

Finally, this dissertation is dedicated to my dear children; Miray Selen TEKÇE, Mehmet Efe TEKÇE, and Umay Ece TEKÇE. They form the very basis of my motivation for the future and new horizons, inspiring and refreshing at every step of this endeavor called life.

Mustafa Serdar TEKÇE

12/01/2024

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with “scientific plagiarism detection program” used by Eskişehir Technical University, and that “it does not have any plagiarism” whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

Mustafa Serdar TEKÇE

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Page

GLOSSARY OF SYMBOLS AND ABBREVIATIONS

A	:	Coefficient Matrice of Systems of Equations
A	:	Denotation for Area
a	:	Discretized Coefficient
a_{ij}	:	An Element of Coefficient Matrice
AMG	:	Algebraic Multigrid
b	:	Constants Vector of Systems of Equations
bc	:	Boundary Condition
c	:	Denotation for Coarse Grid
C	:	Generic Denotation for Cell
cc	:	Convergence Criterion
cD	:	Multiplier Constant of Derivative Term
ce	:	Current Error
CFD	:	Computational Fluid Dynamics
CG	:	Conjugate Gradient
cI	:	Multiplier Constant of Integral Term
cP	:	Multiplier Constant of Proportional Term
cR	:	Current Mean Residual
dc	:	Desired Convergence
Dc	:	Criteria Constant to against Derivative Term
df	:	DataFrame Object
DT	:	Derivative Term
e	:	Error Vector
E, e	:	Denotation for East
f	:	Denotation for Fine Grid
F	:	Finalization
fc	:	Final Convergence
FVM	:	Finite Volume Method
GMG	:	Geometric Multigrid
h	:	Denotation for Grid Resolution Level
HPC	:	High Performance Computing

I	:	Initialization
Ic	:	Criteria Constant to against Integral Term
IT	:	Integral Term
i	:	Cell Indice in the first direction of domain
i	:	Denotation for Cell Number
i_max	:	Maximum Number of Iterations
i_min	:	Minimum Number of Iterations
ic	:	Iteration Count
iR	:	Initial Mean Residual
j	:	Cell Indice in the second direction of domain
k	:	Heat Transfer Coefficient
k	:	Denotation for Process Sequence during an Iteration
L	:	Lower Part of Decomposed Coefficient Matrice
MLAT	:	Multi-Level Adaptive Technique
n	:	Generic Denotation for Number of Item
N, n	:	Denotation for North
NoC	:	Number of Cells
P	:	Prolongation
P	:	Denotation for the Cell being processed
pc	:	Prolongation Criteria
Pc	:	Criteria Constant to against Proportional Term
PDE	:	Partially Differential Equation
pe	:	Previous Error
PID	:	Proportional, Integral, Derivative
pR	:	Previous Mean Residual
PT	:	Proportional Term
q, Q	:	Heat Flux
$\bar{r}$	:	Mean Residual
r	:	Residual Vector
R	:	Mean Residual
R	:	Restriction
rc	:	Restriction Criteria
RWU	:	Reference Work Unit

S	:	Source Term
S	:	Standard Deviation
S, s	:	Denotation for South
SI	:	International System of Units
SOR	:	Successive Over Relaxation
T	:	Temperature
t	:	Time
TDMA	:	Tridiagonal Matrix Algorithm
U	:	Upper Part of Decomposed Coefficient Matrice
V	:	Volume
W, w	:	Denotation for West
WU	:	Work Unit
x	:	Solution Vector
x	:	Location in the first direction of domain
y	:	Intermediate Solution Vector
y	:	Location in the second direction of domain
α	:	A parameter of flexible cycle governing prolongation
β	:	A parameter of flexible cycle governing restriction
Γ	:	Coefficient of Diffusive Property
δ	:	A parameter of scheduled cycle governing restriction
ε	:	A parameter of scheduled cycle governing prolongation
ζ	:	Spatial Location
λ	:	Exponential Decay Constant
φ	:	Physical Property

1. INTRODUCTION

Computational Fluid Dynamics (CFD) have become a basic tool to design, to develop and to assess for any subject related to fluid dynamics and/or mass/heat transfer. Many companies and many institutions develop in-house codes and tools in order to reduce the effort required for any research or design. Commercial CFD software focuses on generic industrial problems, while most of the open-source or academic software focuses on special cases at special context.

Main problem with CFD endeavour is that available computational resource is almost always limited in any situation compared to desired amount, regardless of context and case. Thus, any effort is highly valuable that achieves solution at a desired accuracy while keeping resource (memory, processor, time, power) costs minimal.

Accuracy of CFD simulations mostly depends on the grid resolution of the domain. Assuming convergent solution of a CFD setup is present, increasing resolution adversely effects convergence rate of solution [1]. High resolution representation of domain increases accuracy, requires high resource, increases cost and decreases convergence rate. Low resolution representation of domain decreases accuracy, requires low resource, decreases cost and increases convergence rate. Thus, any CFD campaign aims to determine appropriate domain resolution for given problem through validation and mesh dependency studies [1] before venturing further towards generation of data sets that require high resource allocation for a tremendous amount of time. Compromization between accuracy and cost is crucial for efficiency while using iterative methods to solve the given problem.

Tridiagonal Matrix Algorithm (TDMA) [2], Successive Over Relaxation (SOR) [3] and Conjugate Gradient (CG) [3] can be listed as improvements over simple iterative methods. Adaptive grid refinement techniques can be mentioned as an improvement over using unstructured and non-uniform grids to enhance grid employment [4]. Adaptive grid refinement intelligently alters the grid locally and dynamically during the simulation based on various criteria such as flow gradients, shock waves or regions of particular interest [5]. Partition of the computational domain along with parallelization for High Performance Computing (HPC) are other ways to reduce elapsed time to solution while investing in other computational cost items which are required to solve complex modern problems anyway [6].

Although there are several approaches to reduce computational cost, multigrid methods stand out in terms of cost reduction while keeping solution accuracy [7]. Multigrid methods and algorithms reduce computational cost dramatically [8]. Additionally, multigrid methods do not conflict with most of the other cost reduction approaches, making it a much more versatile and reliable tool while amplifying effectivity. Advanced iterative methods, adaptive grid refinement strategies, partition and parallelization can also be employed while using most of the multigrid methods and algorithms [9]. Essence of the multigrid approach being both simple and profound enables such feat.

Adaptive multigrid methods further improve efficiency while preserving all the other aspects of multigrid approach towards solution. Conception of adaptability of the multigrid methods varies. Some adaptive multigrid methods are about adapting intergrid operations based on algebraic relations present at discretised form of the domain [10]. Some adaptive multigrid methods are about adapting intergrid operations to complex geometries or to exploit geometrical relations while doing so [11]. Some adaptive multigrid methods employ varying decomposition [12] and preconditioning [13] strategies. However, adaptability concept of the thesis is the adaptive multigrid schemes which employs reactive steering between grid resolutions according to specified criteria. The very first adaptive multigrid scheme is called Multi-Level Adaptive Technique (MLAT) [14,15]. While different nomenclature for adaptive multigrid schemes is present such as flexible [16,17] and scheduled [18], all employ essentially same algorithm as MLAT. Surprisingly enough, lack of any other adaptive multigrid scheme is the main motivation of the thesis. Developing a new adaptive multigrid scheme is the main objective of the thesis. Increased robustness and performance of this new scheme compared to present ones are secondary objectives.

1.1. Fundamentals

Multigrid methods are based on the very nature of iterative solutions. In order to describe multigrid methods properly, some fundamental concepts regarding iterative solutions of discretized domains such as grid, residual, error; should be elaborated. Only related highlights are pointed out below, assuming basic knowledge on iterative numerical methods and finite volume method are present [19].

Grid is the discrete representation of a domain at a desired resolution. Domain is divided into finite cells (with vertices) in order to discretise mathematical relations of physical phenomena. Cells of a grid may be at different sizes and shapes while providing topological continuity at cell interfaces. Figure 1.1 is given to visualize examples to structured/unstructured fine/coarse grids. Fine and coarse grid concepts are pretty straightforward which correspond to high and low resolution representations of domain, respectively. Structured grid ends up with coefficient matrices that can be related to geometrical discretization of the domain. Unstructured grids have little or no geometrical relation between coefficient matrices and discretized domain. Both structured and unstructured grids can be processed through varying multigrid methods.

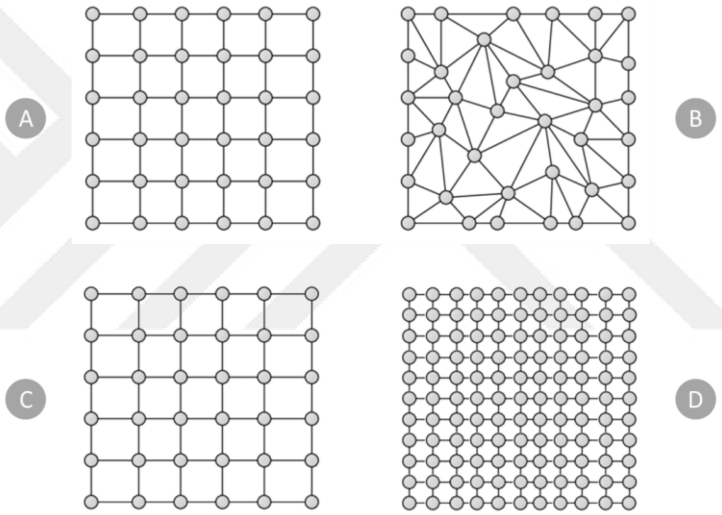


Figure 1.1. *Examples to Structured (A), Unstructured (B), Coarse (C) and Fine (D) grids*

Difference between exact solution and intermediate solution is measured by a concept called residual which in fact is one of the main indicators of convergence when using iterative methods. Residual is the absolute value of differences of a property for grid cells on successive iterations. Mean residual is defined as in Equation 1.2 [1] and converges to zero towards exact solution. “r” is residual, “y” is intermediate solution of a property, “c” is the subscript for “cell”, “i” is the subscript for “iteration”, “n” is the number of cells within given domain.

$$r_c = |(y_c)_i - (y_c)_{i-1}| \tag{1.1}$$

$$\bar{r} = \frac{1}{n}\sum_{c=1}^{n}(r_c) \tag{1.2}$$

Assuming iterative solution exists, residual values should converge towards zero. Residual can be monitored dimensional or non-dimensional. Regardless of dimensionality, appropriate convergence (residual) target is crucial for any iterative solution.

Figure 1.2 visualizes dimensional residual progress of iterative solutions with same convergence criteria for a simple cavity flow on various grid resolutions. Different convergence behaviours and rates are observed for different grid resolutions. More iterations are required for a desired convergence on high resolution grids compared to low resolution grids. Convergence rates decreases with proceeding iterations on all resolutions. Different oscillation behaviours are present on all resolutions as well as different numerical dissipation.

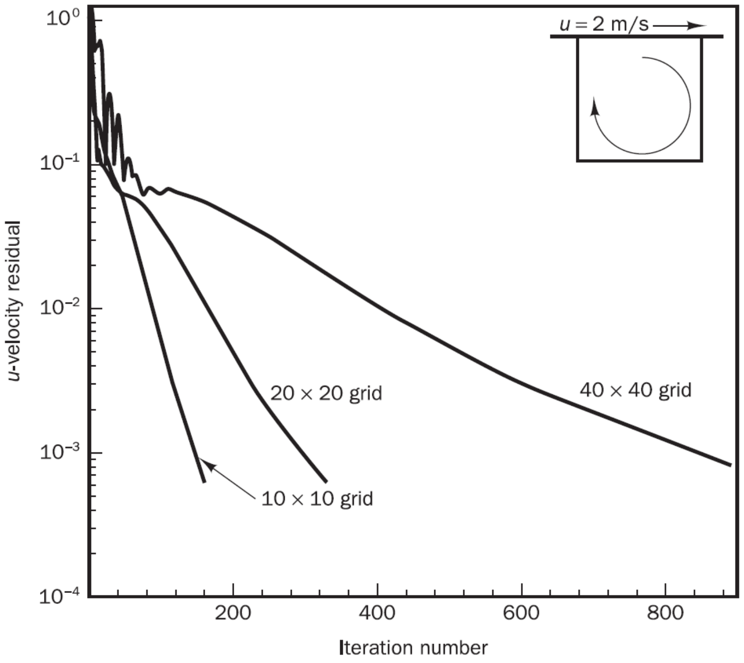


Figure 1.2. *Example of different convergence behaviour of different grids of same domain [1]*

Further mathematical elaboration of residual concept is required in order to establish a proper base for multigrid knowledge. Equation 1.3 defines the solution for a boundary condition problem of a discretized domain. Equation 1.4 defines the relation between intermediate solution and residual. Equation 1.5 defines the error of intermediate solution. Equation 1.6 defines the relation between error and residual. “A” is the coefficient matrices of the system of equations, “x” is the exact solution vector, “b” is the constant vector, “y” is the intermediate solution vector, “r” is the residual vector, “e” is the error vector; for the iterative solution of the boundary condition problem.

$$A.x = b \tag{1.3}$$

$$A.y = b - r \tag{1.4}$$

$$e = x - y \tag{1.5}$$

$$A.e = r \tag{1.6}$$

Equation 1.6 clearly states that error vector and residual vector are directly related. Exact error vector of an iterative solution is always unknown. Thus, residual vector is used in order to check convergence criteria for iterative solutions of discretized domains.

Equation 1.6 also shows that relation between error vector and residual vector depends on coefficient matrices of discretized domain. Thus, relation between error vector and residual vector changes for different grids. Likewise, residual progress through iterations will be different with each different grid that represents the same domain.

It is emphasized that behaviour of error propagation and error dissipation with respect to iterative methods, discretization methods, grid resolution were studied extensively [1]. It is observed that oscillating errors at shorter wavelengths dissipate faster compared to errors oscillating at longer wavelengths [18]. As a focal point of the multigrid methods, it is also observed that; according to algebraic inference, there is a definite relation between grid resolutions and oscillation wavelengths of error through progressing iterations [18]. Simply put; error behaviour and progression are directly related to grid resolution. Practical application (or exploitation) of the relation between iterative solution behaviour and resolution is the very essence of multigrid methods.

Figure 1.3 to Figure 1.5 conceptually visualizes the relation between wavelengths of error and resolution of the solution. Equation 1.7, Equation 1.8 and Equation 1.9 are used for data generation of the example. Equation 1.7 is selected for the example for having both high and low gradients on exact solution. “c” is the input of the function, “x” is the exact solution of the function, “y” is the intermediate solution, “e” is the error of intermediate solution.

$$x(c) = \frac{\ln(c)}{c} \qquad 1 < c < 11 \tag{1.7}$$

$$error = random(-0.1 \to 0.1) \tag{1.8}$$

$$y(c) = x(c) + e \tag{1.9}$$

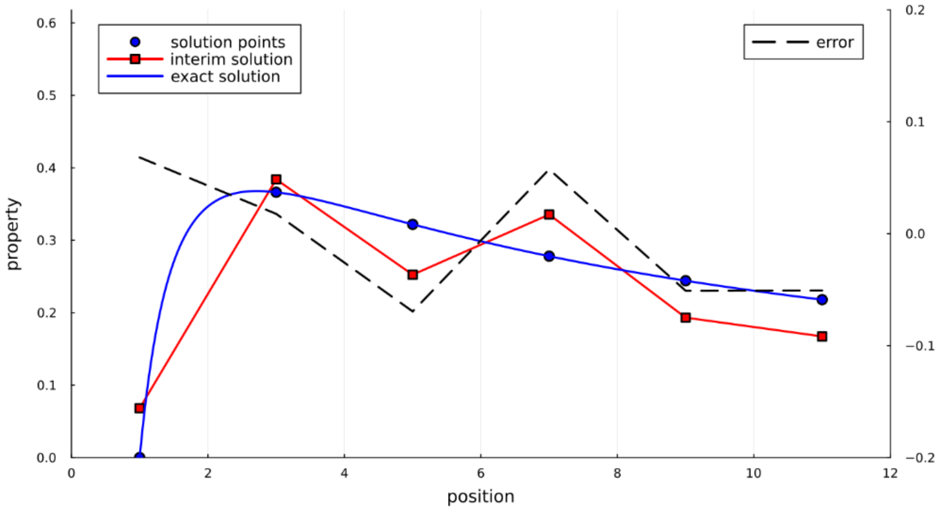


Figure 1.3. *Example to error wavelength at resolution #1*

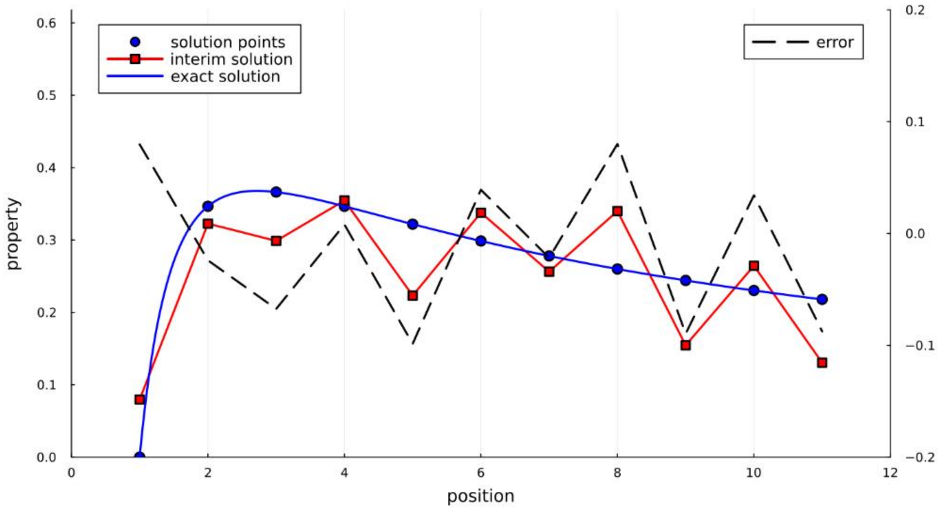


Figure 1.4. *Example to error wavelength at resolution #2*

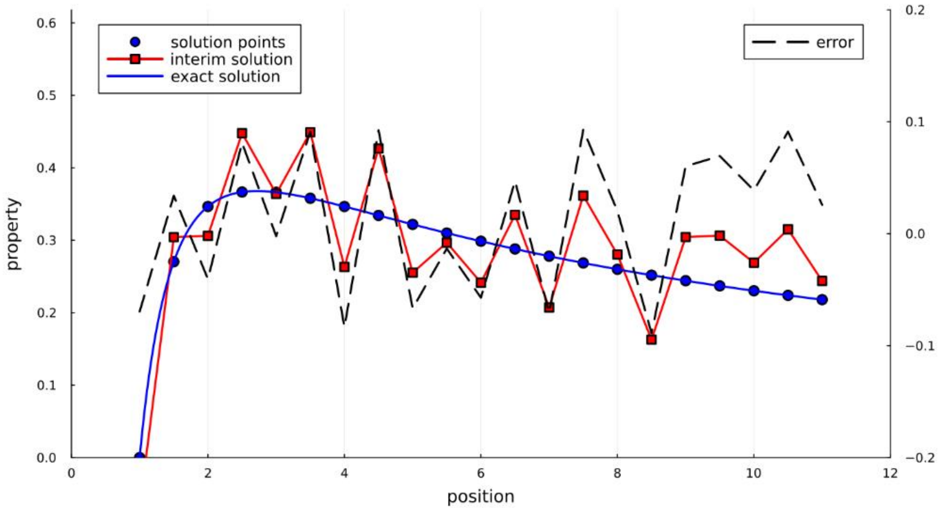


Figure 1.5. *Example to error wavelength at resolution #3*

Examining Figure 1.3 to Figure 1.5, it can be observed that wavelength of the error is directly proportional to grid resolution. Additionally, it can be observed that solving same physical case (dimensions, operating conditions, boundary conditions, etc..) on different resolutions impose different results.

Understanding error wavelength concept is essential; and its mathematical and physical implications are difficult to comprehend at first glance. Since error wavelength is a mathematical phenomenon related to size of the matrices; iterative solution of different coefficient matrices will progress differently. Assuming fine and coarse grid solutions are present, iterative solution progress of fine grid will have shorter wavelengths of errors compared to iterative solution of coarse grid. Vice versa, iterative solution of coarser grid will capture longer wavelength error behaviour. Thus, fine grids would capture and converge occurrences of high gradient changes within domain, while coarse grids would be more efficient on capturing and converging occurrences of low gradient changes within domain. Coarse grids will have mathematically shorter error wavelengths compared to fine grids, while fine grids will have physically shorter error wavelengths compared to coarse grids.

Making use of different error behaviour and computational cost of different grid resolutions for iterative solution of boundary condition problems are the very essence of multigrid methods. With composing proceedings and progresses of different resolutions, multigrid algorithms increase convergence rates while keeping solution accuracy of fine grids.

1.2. Basic Operations

There are several basic operations required for a functioning multigrid cycle/algorithm besides iterative solution operations and finite volume method functions [1]. Each operation or process has a specific purpose to carry out. Main building blocks of any multigrid cycle or algorithm could be listed as initialization, restriction, prolongation and finalization.

Initialization is the preliminary phase of the multigrid cycle between initial guess of the solution and first cycle operation (i.e., restriction). Initialization is basically running the iterative solution starting with initial guess and ending at a specified iteration count, a crude convergence target or an event check. Initialization occurs at initial (or final) resolution which is the finest grid resolution.

Initialization plays an important role for further cycle operations by clearing out irregularities caused by initial guess and boundary conditions on the very first iterations of the solution process. Initial guess of a boundary condition problem is mostly inconsistent with final solution. Boundary conditions may also impose challenging gradients before iterative solution get on track. Consequently, first iterations of iterative solutions highly unstable in terms of residual progress. Clearing out this unstable phase of iterative solution greatly increases viability of upcoming operations.

Restriction is coarsening resolution of the grid and migrating available data accordingly. Restriction is transferring the current state of iterative solution from fine grid to coarse grid. Downgrading resolution of the solution can be processed for both structured and unstructured grids. Restriction of solutions on unstructured grids require algorithms based on algebraic methods. Geometric methods for restriction of solutions requires structured grids.

There are two types of grids regarding domain representation. Vertex centred grids discretise domain around vertices at cell boundaries. Cell centred grids discretise domain around vertices at cell centre. Figure 1.6 illustrates restriction output of a fine grid which is a coarser grid for a structured grid on a two-dimensional domain.

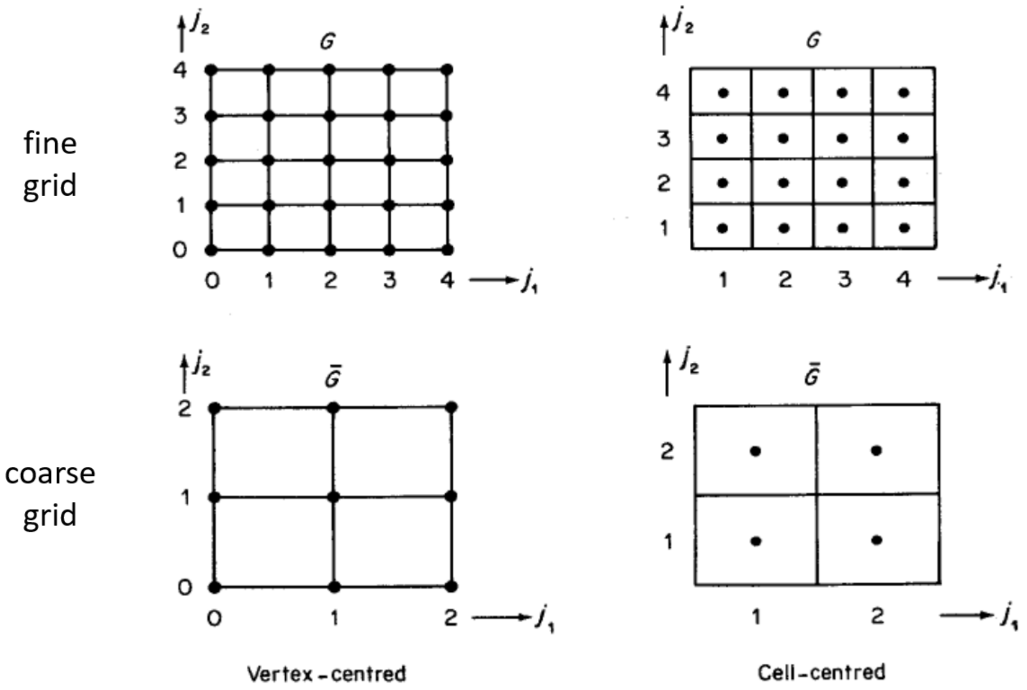


Figure 1.6. *Examples to vertex centered and cell centered grids [10]*

Restriction operation can be done through various approaches. Injection method for restriction is executed by copying some of the values of fine grid to coarser grid and is more suitable for vertex centred grids [18]. Injection method on a vertex centred grid is simply like deleting some of the vertices. Equation 1.10 is given to represent injection method for a one-dimensional restriction. “u” is a physical property of the domain, “i” is the subscript for vertex number (or ID), “c” denotes coarse grid, “f” denotes fine grid.

$$injection: \; u_i^c = u_{2i}^f \tag{1.10}$$

Weighting method for restriction is executed by calculating coarser grid values by interpolating fine grid values and is more suitable for cell centred grids [18]. Equation 1.11 is given to represent weighting method with linear interpolation for a one-dimensional restriction. “u” is a physical property of the domain, “i” is the subscript for vertex number (or ID), “c” denotes coarse grid, “f” denotes fine grid.

$$weighting: \; u_i^c = \frac{1}{2}\left(u_{2i-1}^f + u_{2i}^f\right) \tag{1.11}$$

Figure 1.7 illustrates a one-dimensional example of restriction by weighting method. Figure 1.8 illustrates a two-dimensional example of restriction by weighting method.

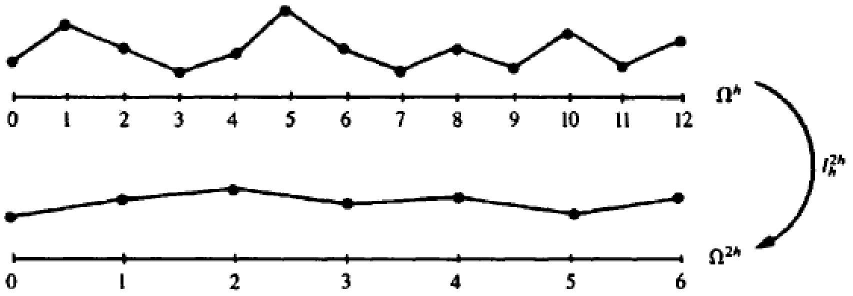


Figure 1.7. *Example to one dimensional restriction operation with weighting method [18]*

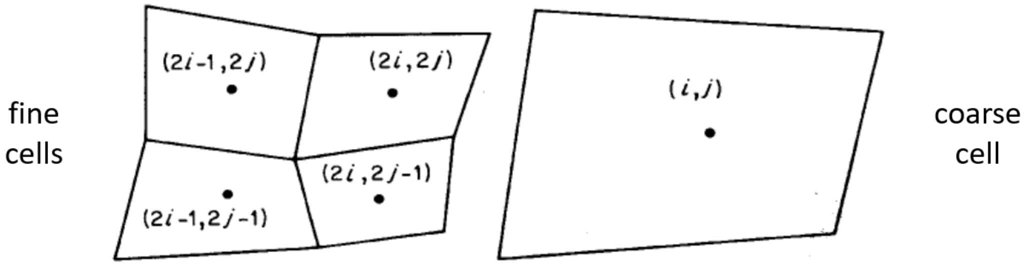


Figure 1.8. *Example to two-dimensional restriction operation with weighting method [18]*

Prolongation is increasing resolution of the grid and migrating available data accordingly. Prolongation is transferring the current state of iterative solution from coarse grid to fine grid. Prolongation is simply the reversed operation of restriction which has more complexity of execution. Upgrading resolution of the solution can be processed for both structured and unstructured grids. Prolongation of solutions on unstructured grids require algorithms based on algebraic methods. Geometric methods for prolongation of solutions requires structured grids.

Prolongation operation can be done through various interpolation methods. Establishing appropriate interpolation method is crucial while respecting boundary conditions of the domain. Equation 1.12 is given to represent linear interpolation for a one-dimensional prolongation. “u” is a physical property of the domain, “i” is the subscript for vertex number (or ID), “c” denotes coarse grid, “f” denotes fine grid. Figure 1.9 illustrates a one-dimensional example of prolongation.

$$u_{2i}^{f} = u_{i}^{c} \qquad u_{2i+1}^{f} = \frac{1}{2}(u_{i}^{c} + u_{i+1}^{c}) \tag{1.12}$$

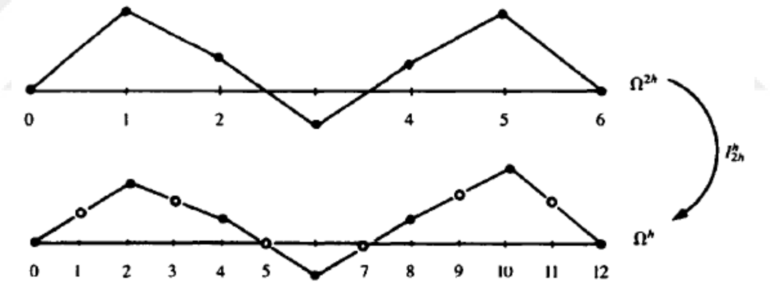


Figure 1.9. *Example to one dimensional prolongation operation with linear interpolation [18]*

Finalization is the carry through phase at the end of the multigrid cycle following last cycle operation (i.e., prolongation). Finalization is basically proceeding the iterative solution delivered through a cycle operation (most probably prolongation) and ending at the final convergence target which will deliver final solution. Finalization occurs at final (or initial) resolution which is the finest grid resolution.

Finalization aims to smooth out the residue of the last cycle operation and to diminish the errors transferred between resolutions. Finalization also aims to continue iterations until desired convergence target is reached. Additionally, finalization phase ensures that final solution is delivered at desired resolution.

1.3. Cycle Descriptions

Multigrid cycles are procedural integration of various grid operations, solution iterations and decision checks composed in a specific algorithm. Various multigrid cycle definitions are present in literature and are used in software. Selecting, defining or setting an appropriate multigrid cycle depends on the case and requires exercise before execution. Multigrid cycles improve convergence rates of iterative solutions, but reckless execution of multigrid cycles may hinder effectivity.

Most of the multigrid cycle definitions have predetermined sequential order of operations and iterations. Figure 1.10 and Figure 1.11 illustrates V cycle and W cycle that are mentioned (without further elaboration) to be members of the μ cycles where μ parameter is 1 and 2 respectively [10]. Figure 1.12 illustrates F cycle that is defined with a specified pattern that progressively increases the top level of fine grid resolutions [18]. Figure 1.13 illustrates full multigrid initialization (FMG) that seems to be defined similar to F cycle without having initial restriction operations and initialized with the coarsest resolution level [18]. "I" denotes initialization, "S" denotes a sweep of iterations, "R" denotes restriction, "P" denotes prolongation, "F" denotes finalization.

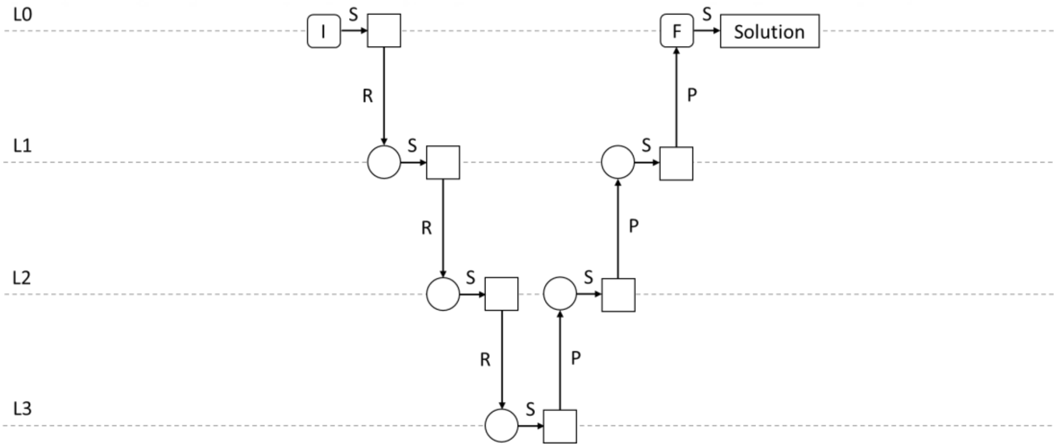


Figure 1.10. *Pattern of a V multigrid cycle with four resolution levels*

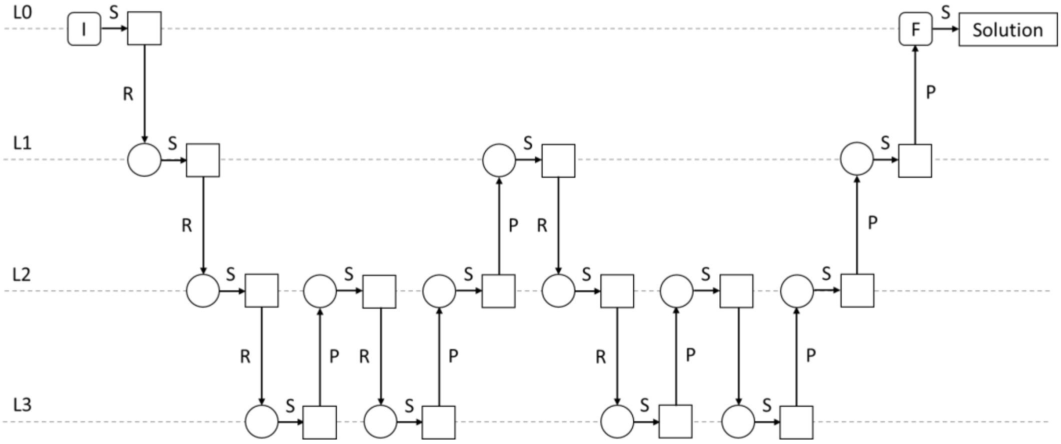


Figure 1.11. *Pattern of a W multigrid cycle with four resolution levels*

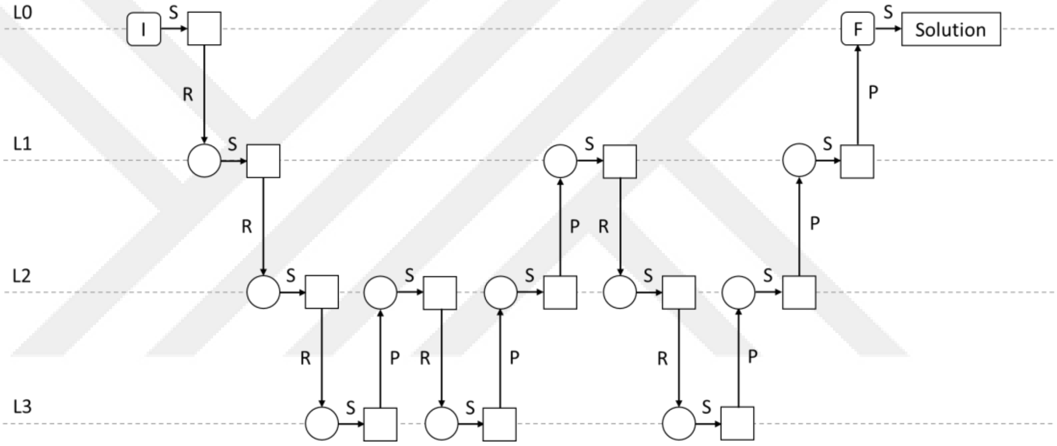


Figure 1.12. *Pattern of a F multigrid cycle with four resolution levels*

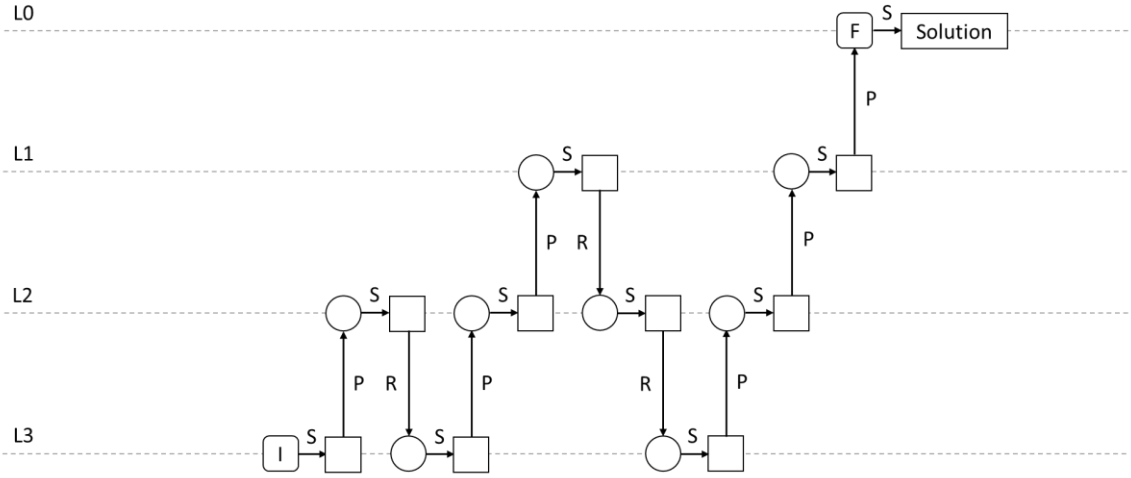


Figure 1.13. *Pattern of a FMG multigrid initialization with four resolution levels*

Theoretically, countless multigrid cycle definitions can be made. Multigrid cycle definition may or may not have a pattern or an algorithm. Regardless of definition, any multigrid cycle should serve a purpose which will most likely be related to case at hand. In some cases, decreasing computational cost without any other considerations may be sufficient. In some cases, multigrid cycles are needed to increase resolution to capture high gradients of properties without increasing computational cost. In some cases, multigrid cycles are beneficial to solve domains with complex geometries at high resolution grid without having to implement grid refinements or further user effort. Whatever the cause, taking the effort to define an appropriate multigrid cycle specific to a case is valid. Defining adaptable multigrid algorithms are another approach to have convenient multigrid cycles for varying cases.

Algorithms which adapt multigrid cycle pattern by automating the execution of restriction and prolongation operations based on decisions around residual progress, are called "flexible" [16,17] in commercial software and "scheduled" [18] in literature. Adaptive multigrid cycles are driven by the conditions of iterative progress. Steering between resolutions without predetermined definition makes the cycle versatile against the ever-changing state of the iterative solution.

Flexible multigrid cycle monitors the residual progress to decide when to execute restriction or prolongation. Restriction operation is decided while iterating solution when residual progress is under an expected convergence rate. Equation 1.13 is used to monitor restriction condition. Prolongation operation is decided while iterating solution when a target residual level which is a proportion of the initial residual level at current resolution, is reached. Equation 1.14 is used to monitor prolongation condition. "α" is the non-dimensional constant of prolongation condition, "β" is the non-dimensional constant of restriction condition, "R" is the mean (non-dimensional) residual, "i" subscript denotes the iteration count, "0" subscript denotes the initial residual. "α" has a default value of 0.3 and "β" has a default value of 0.7 [16].

$$if\ (\ R_i > \beta R_{i-1}\ )\ is\ true\ then\ restrict \tag{1.13}$$

$$if\ (\ R_i < \alpha R_0\ )\ is\ true\ then\ prolong \tag{1.14}$$

Figure 1.14 illustrates an example to the flow chart of a flexible multigrid cycle. Superscript of "R" denotes resolution level, "φ" denotes the property that is being iteratively solved.

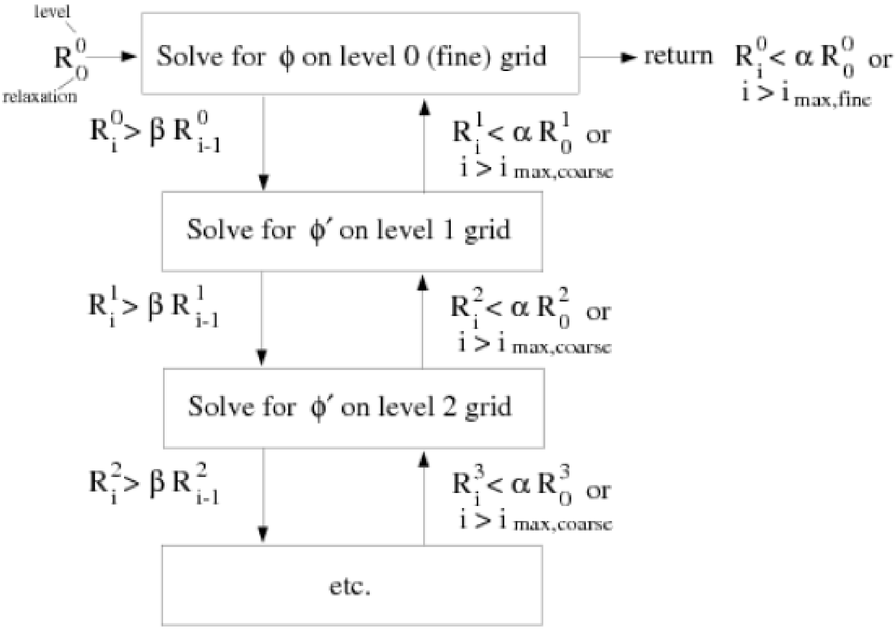

level
R_0^0
relaxation
Solve for ϕ on level 0 (fine) grid
return $R_i^0 < \alpha R_0^0$ or $i > i_{max,fine}$
$R_i^0 > \beta R_{i-1}^0$
$R_i^1 < \alpha R_0^1$ or $i > i_{max,coarse}$
Solve for ϕ' on level 1 grid
$R_i^1 > \beta R_{i-1}^1$
$R_i^2 < \alpha R_0^2$ or $i > i_{max,coarse}$
Solve for ϕ' on level 2 grid
$R_i^2 > \beta R_{i-1}^2$
$R_i^3 < \alpha R_0^3$ or $i > i_{max,coarse}$
etc.

Figure 1.14. *Schematic of flexible multigrid cycle [16]*

Scheduled multigrid cycle monitors the residual progress to decide when to execute restriction or prolongation similarly to flexible multigrid cycle. Restriction operation is decided while iterating solution when residual progress is under an expected convergence rate. Equation 1.15 is used to monitor restriction condition. Prolongation operation is decided while iterating solution when a target residual level (at coarser grid) which is a proportion of the final residual level at previous resolution (at finer grid), is reached. Equation 1.16 is used to monitor prolongation condition. "ε" is the non-dimensional constant of prolongation condition, "δ" is the non-dimensional constant of restriction condition, "R" is the mean (non-dimensional) residual, "c" superscript denotes coarse grid, "f" superscript denotes fine grid, "i" subscript denotes the iteration count.

$$if\ (\ R_i > \delta R_{i-1}\ )\ is\ true\ then\ restrict \tag{1.15}$$

$$if\ (\ R^c < \varepsilon R^f\ )\ is\ true\ then\ prolong \tag{1.16}$$

Even though flexible and scheduled multigrid cycles have slight differences in execution, are conceptually same in essence. Both flexible and scheduled multigrid cycles use non-dimensional constants to monitor operation conditions to steer the cycle towards final solution.

2. LITERATURE

Multigrid is a numerical method to increase convergence rate of the iterative solution of a boundary condition problem of a discretized domain while generating accurate solution [20]. Multigrid methods increase convergence rates by making use of different iterative behaviour of various grid resolutions towards high resolution domain solution [21]. Most of the multigrid cycles have predefined algorithms with certain parameters [21]. Essentially, multigrid algorithms exploit varying benefits of iterating a solution on different (high/low) resolutions in order to get an accurate final solution with reduced computational cost.

Multigrid methods emerged in the late of 20th century and became a fundamental tool of modern CFD software. Figure 2.1 is generated with the numbers returned from the publication search executed with the keywords; “multigrid” and “computational fluid dynamics”; over forty years of time.

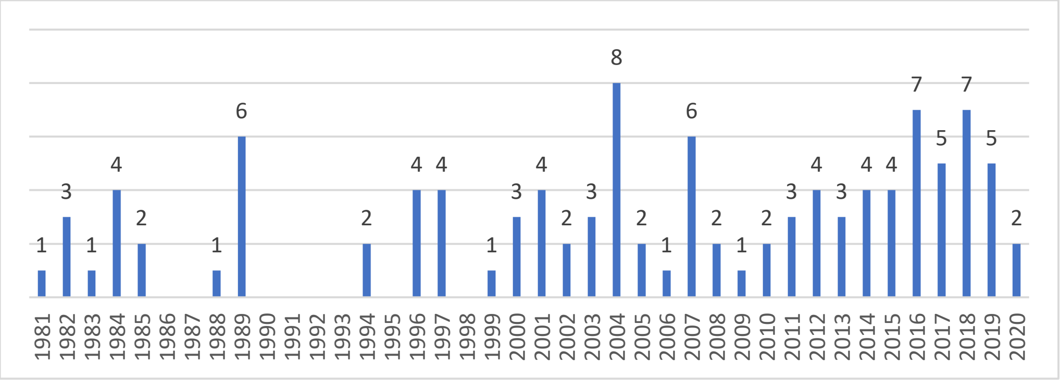


Figure 2.1. *Number of publications about “multigrid” and “CFD” with respect to years*

2.1. Multigrid Methods

The multigrid method is a numerical technique used in computational fluid dynamics (CFD) and other fields of computational science and engineering to solve partial differential equations (PDEs), especially those related to fluid flow. It's a powerful approach for accelerating the convergence of iterative solvers.

Multigrid employs a hierarchy of grids, with each grid having a different level of resolution [10]. These grids can be thought of as a series of meshes, where the finest grid represents the problem's original domain, and coarser grids are obtained by repeatedly coarsening the mesh.

On the finest grid iterative solvers are applied to the discretized PDE to approximate the solution. However, these solvers can be slow to converge, especially when dealing with fine grids. Multigrid introduces a key concept called “smoothing” [18]. Smoothing operations are used to reduce high-frequency error components in the solution. These operations can be relaxation techniques, like Jacobi or Gauss-Seidel, and are applied to the solution to make it smoother.

To transfer information between grids with different resolutions, multigrid employs restriction and prolongation operations [18]. Restriction operators aggregate data from finer grids to coarser grids, while prolongation operators interpolate data from coarser grids to finer grids.

Once the error is reduced on the fine grid using smoothing, the algorithm transfers the problem to a coarser grid. This process continues recursively until the coarsest grid and/or a condition is reached. On the coarsest grid, a direct solver, such as Gaussian elimination, can be applied to solve the problem efficiently due to the reduced size of the system. Any other convenient iterative solver can also be used on the coarsest grid instead of a direct solver.

After solving on the coarsest grid, the algorithm begins a correction phase, which involves prolongating the solution back to finer grids and adding it to the previously smoothed solution [7]. This corrects errors introduced by solving on the coarser grids.

The entire process of transferring between grids, smoothing, and correction is repeated iteratively until a desired level of accuracy is achieved. Multigrid can significantly accelerate the convergence of iterative solvers, reducing the number of iterations required to reach a solution [8].

The power of multigrid lies in its ability to address error components at different scales efficiently [18]. High-frequency error components are more effectively treated on finer grids, while low-frequency error components are addressed on coarser grids. This hierarchical approach can dramatically improve the convergence rate, making it a popular choice for solving complex CFD problems with fine grids, where other solvers may struggle to converge in a reasonable amount of time [8].

Multigrid methods can be divided into two categories considering respective approaches to grid interpretation. Algebraic multigrid methods (AMG) focus on operations regarding matrix of coefficients which does not require any geometrical constraint [22]. On the other hand, geometric multigrid methods (GMG) focus on operations regarding geometrical representation of the domain which does require some form of relationship between grid and matrix of coefficients [22]. Both AMG and GMG have unique feats which can be a pro or a con depending on the situation and/or case.

Algebraic multigrid methods offer a distinctive approach unlike geometric multigrid methods which rely on the grid's geometric structure. Algebraic multigrid methods do not require detailed information about the grid or its physical properties. Instead, they operate based on the algebraic relationships between grid points.

Algebraic multigrid methods have garnered significant attention due to their ability to handle unstructured grids, complex geometries, and irregular mesh structures. The distinguishing feature of these methods is that they build a hierarchy of grids solely based on the algebraic connectivity between grid points.

Large system of algebraic equations represents the discretized equations and involves a matrix of coefficients and a vector of unknowns. The matrix structure may or may not reveal any obvious geometric hierarchy. Algebraic multigrid methods, through a systematic approach, construct a multilevel grid structure without explicitly considering grid points, making them highly adaptable to a wide range of grid configurations [22].

Geometric multigrid methods, in contrast to algebraic methods, leverage the geometric structure of the grid to take advantage of the grid's hierarchy, where a coarse grid is embedded within a finer grid, and each level represents a different level of resolution.

Geometric multigrid methods typically start with an initial guess for the solution and apply relaxation techniques, such as Gauss-Seidel or SOR, to smooth out errors at each level. As the iterations progress, the grid hierarchy allows for the transfer of information between different resolutions. This interaction between grid levels helps correct errors and improve the accuracy of the solution.

Geometric multigrid methods excel in situations where the grid has a well-defined geometric hierarchy, such as structured grids [22]. They are particularly effective for problems with regular geometries and highly isotropic flow conditions.

Overall, the choice between algebraic and geometric multigrid methods depends on the nature of the problem and the grid type. Algebraic multigrid methods offer flexibility in handling irregular grid structures, while geometric multigrid methods are well-suited for problems with well-defined grid hierarchies [22].

2.2. Multigrid Schemes

Multigrid schemes are a category of methods employed in CFD to accelerate convergence in iterative solvers. Fixed-pattern multigrid schemes, such as the V-cycle, W-cycle, and F-cycle, are widely used and have demonstrated their effectiveness in improving convergence speed [7].

In a multigrid cycle, the grid is successively refined and coarsened in a consequential manner. This process involves moving between different levels of grid resolution. At each level, a relaxation method, often a simple iterative technique like Gauss-Seidel, is applied to smooth out errors in the solution. The process of moving between grid levels allows for the transfer of information, helping to correct errors more effectively. The W-cycle and F-cycle are variations of the V-cycle, introducing more coarsening and refinement steps in the multigrid scheme. These fixed-pattern multigrid schemes aim to address issues with convergence by taking advantage of grid hierarchy and the interactions between different resolutions.

Fixed-pattern multigrid schemes are robust and well-established, making them prior choice for many CFD simulations [1]. However, they may not always be the most efficient option in all scenarios. For some complex problems, particularly those involving irregular geometries or highly anisotropic grids, adaptability in the multigrid cycle is essential.

Adaptive multigrid schemes take the principles of fixed-pattern multigrid a step further by introducing adaptability into the process. These schemes dynamically adjust the multigrid cycle based on the problem's characteristics [17].

The adaptive multigrid approach recognizes that not all regions of the flow field require the same level of attention. In areas with smooth and well-behaved flow, coarser grid levels and fewer iterations suffice. On the other hand, in regions with complex turbulence or steep gradients, finer grids and more iterations are necessary for accurate convergence.

To achieve adaptability, adaptive multigrid schemes use various criteria to determine when to refine or coarsen the grid, such as error levels, gradient steepness, or flow behaviour [8]. This adaptability enhances convergence speed by focusing computational effort where it is most needed. It effectively balances the need for accuracy with computational efficiency, making it a powerful tool in addressing a broad range of flow problems, including those with dynamic or changing flow patterns.

Adaptive multigrid schemes are highly versatile and are especially beneficial for problems where flow conditions vary significantly throughout the domain. They can adapt to handle both complex and relatively straightforward flow regions, optimizing the convergence process and ensuring that CFD simulations produce accurate and reliable results [8].

2.3. Recent Studies

Some of the recent studies in multigrid methods have concentrated on enhancing the efficiency and reliability of these techniques. One prominent area of research has been the adaptation of multigrid methods to high-performance computing (HPC) environments. As computational resources continue to evolve, multigrid methods need to be optimized for parallel computing platforms, enabling the simulation of more complex and realistic scenarios. The scalability of multigrid methods is crucial, especially for problems that require massive computational power, such as weather forecasting, turbulent flow simulations, or aerodynamic analysis of intricate aircraft designs. Recent research has focused on developing parallelization strategies and hybrid solvers that harness the power of modern HPC architectures.

Parallel multigrid methods have several advantages over sequential ones, including faster computation, scalability, reduced communication overhead, and efficient implementation. Parallel multigrid methods can solve large problems faster than sequential methods by distributing the computation across multiple processors [23,24,25]. Parallel multigrid methods can reduce communication overhead by using grid partitioning and ensuring that time spent on communication is maintained at a small fraction of computation time [26,27]. Parallel multigrid methods can be implemented efficiently on parallel machines, and many research projects have been conducted on parallel multigrid methods, including parallel geometric multigrid methods [28]; addressing a variety of subjects from proposed new algorithms to theoretical studies to questions about practical implementation [25].

Parallelization of multigrid methods is an important area of research for solving large computational problems on high performance, massively parallel computers. There are various parallel implementations of multigrid methods, and some components of the classical algorithm require new parallel approaches.

A new parallel algorithm [29] for the multigrid process has been developed, including a modification based on a V-cycle mixed with the two-grid method. With this modification, parallel computing is enabled for each grid resolution. Convergence rate of the initial value problem is enhanced by implementing parallelization in post-smoothing while keeping sequential pre-smoothing.

In a parallel multigrid method [30], load balancing, adaptive grid refinement and eventually solution of PDEs on the processed grids are the main phases of the method. Defining domain partition and show nodes are initial and essential for further operations. Compatible adaptive grid refinement succeeds domain definition. Balanced parallelization of the partitioned grids is carried out to various metrics of partitions and relations between them. Hierarchical multigrid methods then used to achieve high parallel efficiency on small cluster computers.

Multigrid methods have proved to be among the fastest numerical methods for solving a broad class of problems, from many types of partial differential equations to inverse problems [24]. Reviews of various sequential and parallel methods conclude with that multigrid methods will preserve their contribution over iterative solutions, even if parallel computing takes over serial computing [24].

Algebraic multigrid (AMG) is a very efficient algorithm for solving large problems on unstructured grids. While much of it can be parallelized in a straightforward way, some components of the classical algorithm, particularly the coarsening process and some of the most efficient smoothers, are highly sequential, and require new parallel approaches. One of the algebraic approaches is based on decomposition of domain into a number of sub-domains (partitions) with an overlap [23]. Various parallel coarsening schemes, interpolation procedures, parallel smoothers are reviewed [31]. Vast number of research of parallel AMG methods are present [31]. One of the approaches is seemed standing out with the concept of compatible relaxation [32].

Advancements in multigrid methods have also targeted to improve the handling of complex geometries. Problems involving irregular or unstructured grids have historically posed challenges for multigrid solvers. Researchers have developed techniques to adapt multigrid methods to effectively handle these complex geometries, ensuring that they remain a valuable tool for a wide range of CFD applications.

Some of the recent studies regarding geometric multigrid methods (GMG) [33] have focused on the adaptability of multigrid methods to complex geometries. A meshless geometric multigrid method has been proposed for complex geometries with an improved cell coarsening algorithm [34]. A flexible, parallel, adaptive geometric multigrid method has also been developed for adaptively refined meshes for massively parallel computations [35]. Additionally, an adaptive geometric multigrid method has been developed for the mixed finite cell formulation of Navier-Stokes equations [36]. Results from these studies indicate that the presented multigrid methods are capable of solving the model problems independently of the problem size and are robust [34,36].

Research on multigrid schemes has focused on refining existing approaches and introducing novel ones to address a range of flow problems effectively. Fixed-pattern multigrid schemes have undergone further investigation to optimize their parameters for improved convergence. The challenge with fixed-pattern schemes is that they may not always provide the fastest convergence in every situation. Recent research has delved into the fine-tuning of parameters, such as the number of grid levels and relaxation factors, to adapt multigrid schemes for specific problems. By optimizing these parameters, researchers aim to enhance convergence speed and efficiency in simulations across a diverse set of flow conditions and geometries.

In practice, tuning the parameters of multigrid methods is a "minimax" problem, minimizing with respect to solver parameters the appropriate measure of work, which involves minimizing an estimate of the spectral radius of a stationary iteration, or the condition number of a preconditioned system, in terms of a symbol representation of the algorithm [37].

There have been some recent developments in the field of parameter-dependent multigrid methods. A parameter-dependent smoother for the multigrid method has been proposed, which uses tensor formats to efficiently solve parameter-dependent linear systems [38]. The integers ν1 and ν2 are parameters in the scheme that control the number of V-cycles and the number of pre- and post-smoothing steps, respectively [10].

In summary, tuning the parameters of multigrid schemes is important to optimize their performance. Local Fourier analysis is a useful tool for predicting and analysing the performance of multigrid methods. Recent developments in the field of parameter-dependent multigrid methods have proposed a parameter-dependent smoother for the multigrid method, which uses tensor formats to efficiently solve parameter-dependent linear systems.

Adaptability within multigrid schemes has also garnered substantial attention. Studies have explored ways to make multigrid cycles more responsive to the changing flow conditions throughout the simulation. Adaptive multigrid schemes, as mentioned earlier, dynamically adjust the grid based on error levels, gradient steepness, or flow behaviour. These schemes have the potential to significantly reduce the number of iterations required for convergence and ensure that simulations produce accurate results in scenarios with fluctuating flow patterns.

The adaptability of multigrid cycles has been an essential point of research. Researchers have explored ways to automate the selection of grid levels and determine when to refine or coarsen the grid during the simulation. This adaptability is particularly critical for problems with high gradients and/or transient phenomena, where the flow features change dramatically and/or over time.

Multi-level adaptive technique (MLAT) is a combination of multigrid and adaptive methods [14]. The basic idea of MLAT is to work not only on a single grid but on a sequence of grids, each of which is a successor of the previous one. The solution can also be computed on the coarsest grid and then refined successively on finer grids until the desired accuracy is achieved [14].

The MLAT algorithm uses a combination of relaxation sweeps and intergrid operations to solve boundary value problems [14]. MLAT is a challenging technique to implement because its basic software components are strongly interrelated, and thus the modularization is nontrivial [15]. Program design is further complicated by efficiency considerations [15]. However, the abstract mesh concept has been developed to address these issues and its implementation in a patch-adaptive multigrid program has been successful [39].

MLAT offers several advantages over traditional numerical methods, including improved efficiency, better accuracy, adaptability, and flexibility. These advantages make it a powerful tool for solving partial differential equations in a wide range of applications.

MLAT improves the efficiency of numerical solutions to PDEs by adapting the discretization to the local behaviour of the solution [15]. This means that the algorithm can use a coarser grid where the solution is smooth and a finer grid where the solution is more complex, resulting in faster and more accurate solutions [15].

MLAT can also achieve higher accuracy than traditional numerical methods by using a sequence of appropriate discretizations [15]. This allows the algorithm to capture the local behaviour of the solution more accurately, resulting in more accurate solutions [15].

MLAT is adaptable to a wide range of problems, including linear and nonlinear, elliptic and mixed-type problems [15]. This makes it a versatile technique that can be used in many different applications.

Since MLAT is a combination of multigrid and adaptive methods, it can be used with different types of discretization methods, including finite-difference and finite-element methods [15]. This makes it a flexible technique that can be adapted to different types of problems.

While MLAT offers several advantages over traditional numerical methods, it also has some limitations. These include complexity, mesh generation challenges and implementation difficulties.

MLAT is a complex technique that requires a significant number of computational and theoretical expertise to implement [15]. The algorithm uses a sequence of increasingly finer discretizations, which may require recursive and/or nested algorithms [15].

MLAT requires the generation of a sequence of meshes, which can be a challenging task [15]. The meshes must be generated in a way that accurately captures the local behaviour of the solution, which can be difficult to achieve [15].

2.4. Summary of Literature

Advanced iterative methods are not applicable to all situations. Adaptive grid refinement mainly aims to increase solution accuracy not convergence rate. Algebraic multigrid methods are mostly concerned with intergrid operation robustness. Geometric multigrid methods are mostly case specific and aim to exploit geometrical relations while transferring data between grid resolutions. Fixed-pattern multigrid schemes are robust but lack efficiency when left without user input.

Studies on multigrid methods are mostly focused on scalability of parallelization on HPCs, adaptation to complex geometries. Studies on multigrid schemes are mostly focused on parameter optimization of fixed-pattern schemes, tuning adaptive algebraic multigrid methods and hybrid schemes with different approaches to domain decomposition and preconditioning.

Although there are various efforts to improve existing multigrid methods as well as proposing new ones occasionally, studies specifically on adaptive multigrid schemes remain fundamental and are simply lacking. Flexible and/or scheduled adaptive schemes are merely a different expression of MLAT while offering no alternative to essence of MLAT as it emerged. Thus, effort of generating a new adaptive multigrid scheme algorithm is considered worthwhile.

3. METHOD

Definitive exploration of adaptive multigrid schemes requires controlled experiments. In order to perform controlled experiments, clarity of parametric models and reference case are also needed. In this manner, thesis is built upon a code from scratch which is consisted of functions with scalable infrastructure. Functions are generated to work with varying inputs and to return output in a definitive format. Output of a function is input of another function and vice versa. Scalable infrastructure that keeps input and output intact, enabled changing the internal sequence/procedure of the function without hindering the main stream of code execution.

Functions and algorithms are presented with input, sequence/procedure and output alongside assumptions and theory of each procedure. Consequently, method of the thesis is presented with both theory and practice in an integrated fashion. Figure 3.1 illustrates the presentation schematic of functions and algorithms.

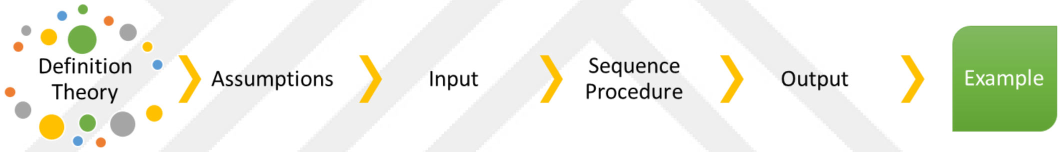


Figure 3.1. *Presentation schematic of functions and algorithms*

All codes of functions/algorithms are shared in Appendices #1 to #4 in the presented order. With careful treatment, all functions/algorithms can be altered to work with another case. Future work presents essential pointers and insightful considerations towards further development of the code.

3.1. Theory

Theoretical basic knowledge to some concepts used in methods are given in this section. These concepts and subjects which are not used directly as they are but are seen as inspiration. Work unit concept is an inspiration to reference work unit parameter which is used to compare cost of multigrid cycles. PID controller theory is an inspiration to steering algorithm of the adaptive cycle that is generated and presented with the thesis. Exponential decay theory is an inspiration to understand residual progress as well as to specify a criterion to the steering mechanism of the adaptive multigrid cycle.

3.1.1. Work Unit

Work unit (WU) is defined as the memory cost of performing one relaxation sweep on the finest grid [18]. It is also noted that intergrid transfer operations (restriction and prolongation operations throughout the cycle) cost 10-20% of the entire cycle [18]. By solving geometric series of the memory costs at each resolution level, a single V multigrid cycle cost is given as 4 WUs for one-dimensional problem, 8/3 WUs for two-dimensional problem and 16/7 WUs for three-dimensional problem [18]. Memory cost of multigrid cycles decreases with increasing dimension of the problem. Figure 3.2 illustrates the memory cost accumulation of a V cycle with four resolution levels; based on solution and right-side vectors [18]. “(n)h” represents (n)th resolution level, “v array” and “f array” represents the solution and right-side vectors respectively.

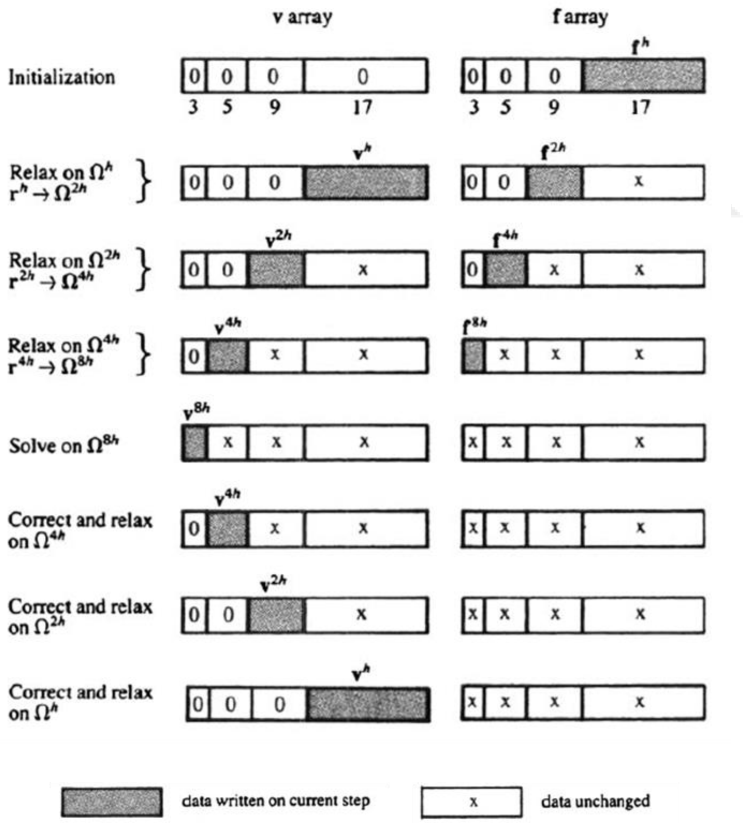


Figure 3.2. *Memory cost diagram of a V cycle with four resolution levels [18]*

Work unit concept as described above and illustrated in Figure 3.2 is ambiguous because memory allocation of any array or vector is dependent to the environment (language, compiler, IDE, operating system, etc…) as well as definition of the array/vector itself. Additionally, most expensive object of an iterative solution is most likely the coefficient matrices rather than solution vector if sparse array infrastructure is not used. Even if sparse arrays are used to store data, matrices and vectors would have varying memory costs depending on the boundary conditions and solver models. Fair comparison of costs of multigrid cycles requires normalized or independent variable as an indicator. Time cost would inherently include any varying parameter of the environment, algorithm and system that multigrid cycle works on.

Work unit concept as a cost indicator is also not comprehensive enough for present applications. Computational cost includes process time in addition to memory allocation. Even though memory cost can be a limitation to an iterative solution at certain situations, time cost is much more significant when considered within the context of modern fast paced business and scientific environment. Time cost inherently includes the processor time, as well as access time to data on memory. Time cost also includes the energy cost to power the computer system during the process.

Measuring computational cost in terms of elapsed time is seen as much more convenient and comprehensive for comparing performance of multigrid cycles. Thus, reference work unit is defined as the elapsed time of performing one iteration on the finest grid.

3.1.2. PID Controller

Proportional-integral-derivative (PID) controller is defined as a control loop which implements feedback to the process in order to achieve continuously modulated control towards a desired state [40]. PID controller continuously calculates an error between a desired and measured state of a process, consequently applies a correction response generated with the combination of proportional, integral and derivative terms [40]. Figure 3.3 illustrates the block diagram of a PID controller feedback loop. Error value (e(t)) is calculated by subtracting measured process variable (y(t)) from desired setpoint (r(t)). Control variable (u(t)) is calculated by combining contributions of proportional (P), integral (I) and derivative (D) terms. Proportional term is calculated by multiplication of proportional gain factor (K_P) and error value. Integral term is calculated by multiplication of integral gain factor (K_I) and integration of error value through the progress. Derivative term is calculated by multiplication of derivative gain factor (K_D) and rate of change of error value through the progress.

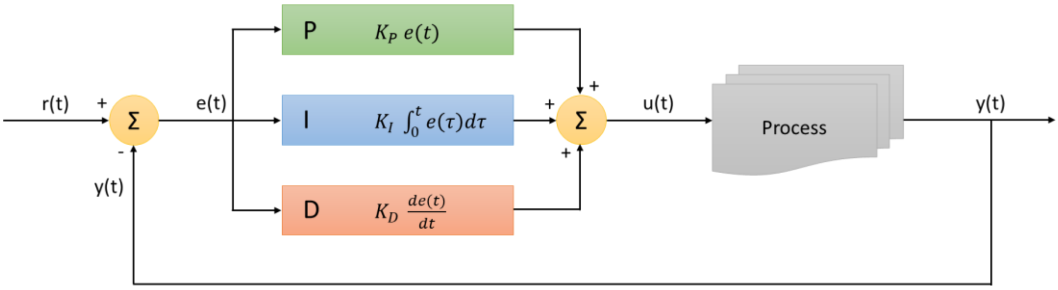


Figure 3.3. *Block diagram of a PID controller feedback loop*

Control variable (u(t)) is generated by adding proportional, integral and derivative terms for an unmodified PID controller. Proportional control variable is proportional to existing error. If there is an error factor having different unit than setpoint and process variable, proportional control fails. Integral control variable is set regarding the persistence of the error. Integral control shows slow reaction at first but shows increasingly intense reaction if error persists. Integral control eliminates offset errors, may induce oscillations in form of constant, growing or decaying sinusoids. Derivative control variable is set regarding the rate of change of the error. Derivative control cannot bring system to setpoint by itself, while flattening the error trajectory towards setpoint. PID controller combines pros and cons of each individual controllers into a much more reliable and adaptable feedback loop.

Although being a widely used and an adaptable feedback loop, PID controller can not be used as described above for steering an adaptive multigrid cycle. Control variable and process variable are simply different in terms of both units and concept for a multigrid cycle.

Setpoint or desired state (r(t)) of an iterative solution is convergence target which is in the units of residual. Consequently, process variable (y(t)) should inherit the same unit in order to subtract one from another. So, setpoint is recognized as convergence target while process variable is recognized as current residual. Therefore, error is recognized the difference between current residual and convergence target.

Steering (restrict, prolong) or termination command is recognized as the input of the adaptive multigrid cycle while output of the multigrid cycle is recognized as the iterative solution of the problem which is monitored through residual.

Thus, proportional-integral-derivative (PID) driven adaptive multigrid cycle is envisioned as the innovative aspect of the thesis based on the inspiration from PID controllers.

3.1.3. Exponential Decay

Exponential decay is defined as a proportional decrease rate of a quantity to its current value over time [41]. Representation of exponential decay in the form of differential equation is given in Equation 3.1 and solution of the differential form in Equation 3.2 for further use. Figure 3.4 contains the exponential decay curves plotted with varying lambda values with a certain (N_0=1) initial value.

$$\frac{dN(t)}{dt} = -\lambda N(t) \tag{3.1}$$

$$N(t) = N_0 e^{-\lambda t} \tag{3.2}$$

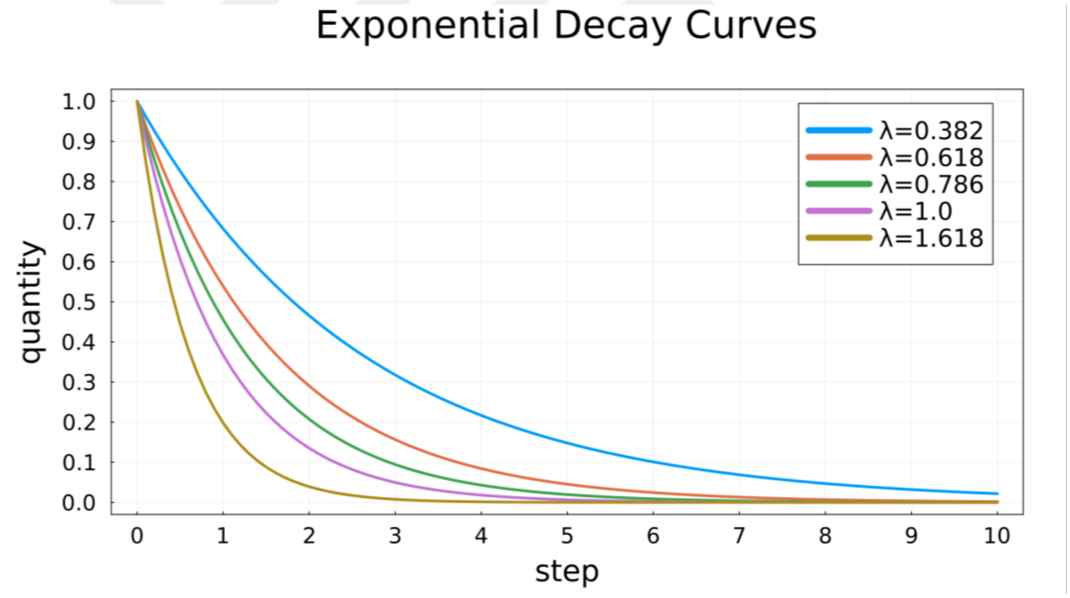


Figure 3.4. *Example to exponential decay curves*

Residual progress curves of convergent iterative solutions are strikingly similar to exponential decay curves. Figure 3.5 gives an example to progress curves of mean residual for different resolutions. Consideration of the similarity between Figure 3.4 and Figure 3.5 gives the impression that exponential decay concept may be useful for the efforts of generation of an adaptive multigrid cycle. Logarithmic behaviour of residual curves on Figure 1.2 enhances the impression to consider exponential decay concept as an asset towards generation of an adaptive multigrid cycle.

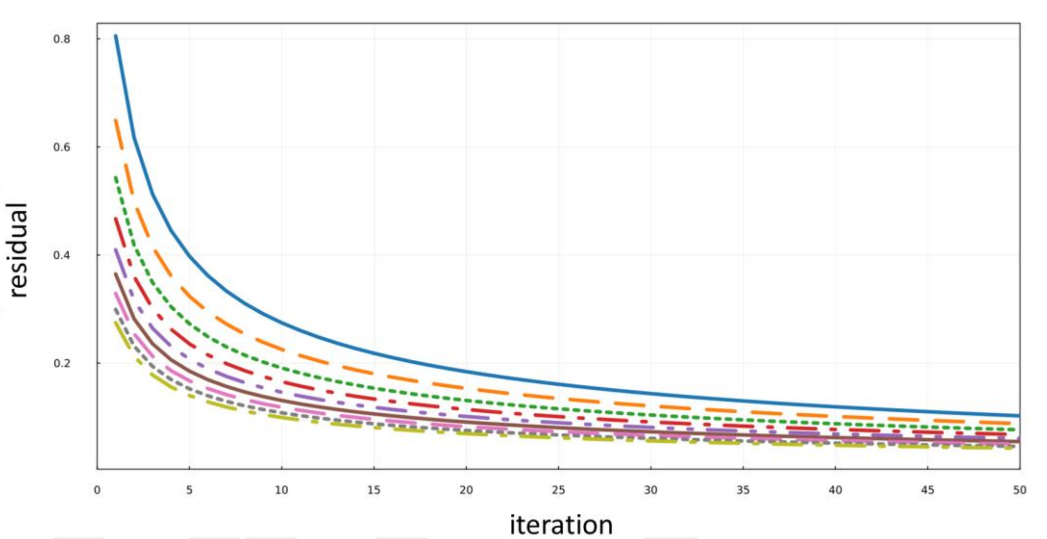


Figure 3.5. *Example to progress curves of mean residual for different resolutions*

Further investigation on exponential decay curves lead to observations on integral and derivative behaviours of the curves. Derivative behaviour is straightforward enough to understand with a constant proportional change as the definition goes. However, inspecting integral behaviour of the curves reveal an interesting aspect of the exponential decay. Figure 3.6 illustrates the integral curves of the exponential decay. All of the integral curves plotted with different lambdas eventually and asymptotically converges to a certain value which in fact is 1/λ. So; for a known or a desired rate of change of decay, there is an integral value that decay can never reach. Vice versa; for a certain integral target value, there should be a minimum decay constant value to reach that integral. In other words, exponential decay behaviour can be used to check if residual progress complies with a desired rate of change; or to check if residual progress exceeded a certain investment (integral) level.

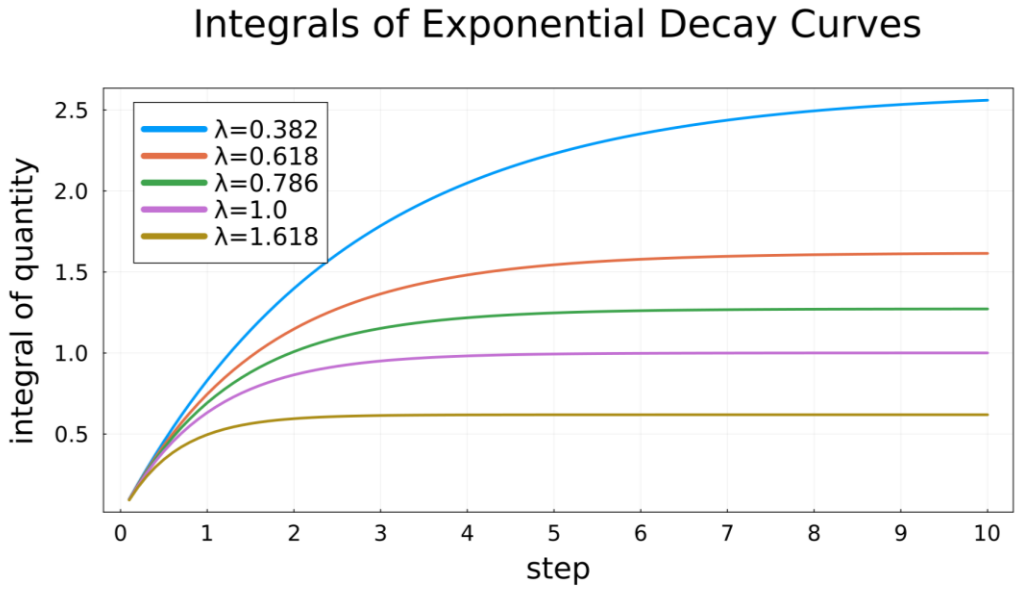


Figure 3.6. *Example to integral of exponential decay curves*

Thus, exponential decay behaviour is considered to understand residual progress and to specify steering criteria of the adaptive multigrid cycle.

3.2. Scope

Exploration space to cover every aspect of an adaptive multigrid cycle definition for the iterative solution of a boundary condition problem using finite volume method is simply vast. Exploration with commercial software is not viable due the simple fact that commercial software is not open-source and does not allow to change most of the parameters. Exploration with open-source software is also not viable due to the complexity of available CFD solvers which will break the focal point of the exploration by forcing a high effort for proper validation. Most sensible approach to exploration was seen as the generation of a simple finite volume method solver and building multigrid algorithms over the definite infrastructure. Generation/development of a code/algorithm from scratch enabled full authority over the controlled experiments of parameters as well as clarity over validation. Exploration without any infrastructural uncertainty improved the merits of the thesis.

Main concern of the thesis is to discover the fundamentals of adaptive multigrid cycles while establishing a viable and valid execution base. Thus, efforts are focused on generation/development of multigrid algorithms rather than generation/development of a CFD solver.

Boundary condition problem is assumed as a two-dimensional diffusion. Thermal diffusion problem on a homogenous plate is selected for the reference case which may have constant temperature and constant heat flux at boundaries. Thermal coefficients are assumed to be constant within the homogenous plate. Another reason to select thermal diffusion case is having analytical Laplace solution which inherently enabling proper validation of results.

Domain is assumed to be represented with a structured grid of uniform elements. Structured grid enabled to use geometric multigrid methods while easing the implementation of finite volume method operations.

Geometric multigrid methods are assumed in order to ease the implementation of restriction and prolongation and to focus on multigrid cycle algorithms.

Central differencing is assumed for discretization which is adequate for the reference case and is convenient to implement on structured grids and to work with geometric multigrid methods.

Gauss Seidel method without any modification is assumed for iterating the solution which is adequate for the reference case.

All variables are in default SI units [42] unless otherwise specified.

3.3. Environment

Selection of the medium for the tool generation is the initial step to consider. Environment should fit to the scope and purpose of the code. Capable of meeting the infrastructure requirements; an object-oriented, functional, fast, high-level, open-source language is considered for tool development. Although "Python" has been the most popular language for software development in recent years, "Fortran" or "C" can still be preferred due to its speed in many engineering applications. Although languages such as "MATLAB" and "VBA" are also widely used, they are not open-source. “The Julia programming language” has been preferred with the claim of being both high-level and fast. According to shared benchmark data, Julia has proven to be as fast as Fortran, even though it is as high-level as Python [http-1]. Figure 3.7 summarizes the results of various languages for various benchmark types.

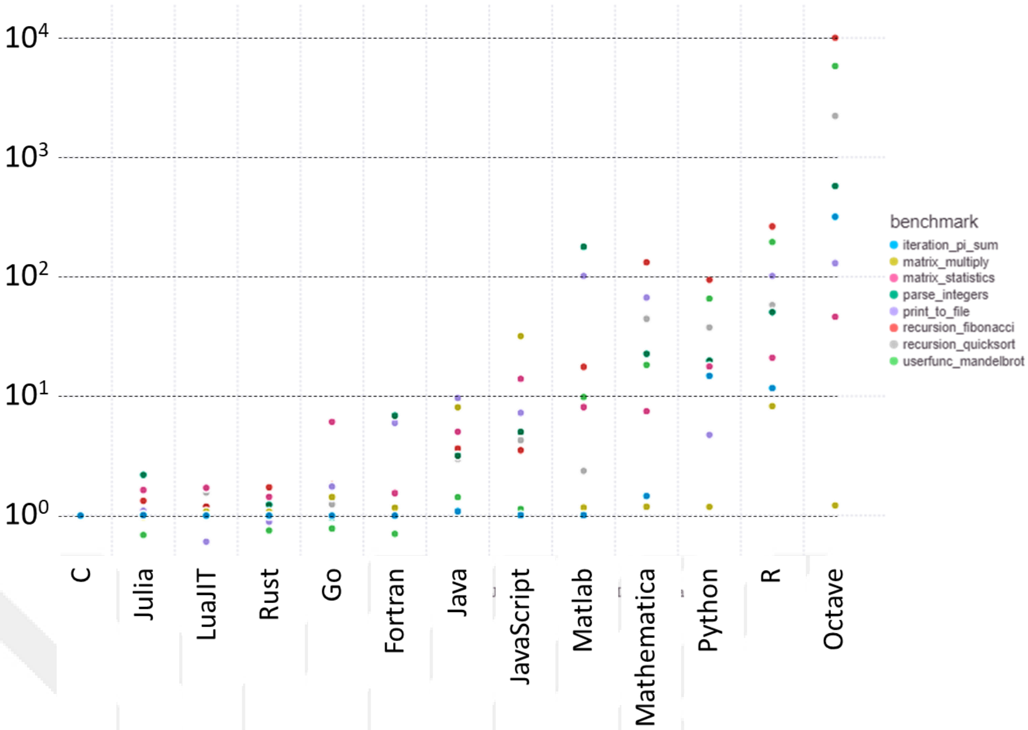


Figure 3.7. *Benchmark results of various programming languages compared to C language [http-1]*

Selection of the language led to specify which libraries to use while keeping tool development authentic as possible. Scope of the thesis focuses on the effectivity of multigrid methods and algorithms. Thus, finite volume methods, iterative solvers and multigrid operations should be performed with authentic functions. Complimentary or basic infrastructure such as; mathematical functions, data base environment, graph plotting, file input output protocols, benchmark tools, are implemented by standard or imported libraries. Comprehensive documentation of the language and standard libraries are available [http-2, http-3, http-4, http-5, http-6]. Documentation of external libraries [http-7, http-8, http-9, http-10] are also available at respective source sites. Table 3.1 summarizes brief explanations to the imported libraries that is selected for code infrastructure.

Table 3.1. *Selected libraries for code infrastructure*

Library	Origin	Explanation
Dates	Standard	Provides functions that generates date and information in various/custom formats.
Printf	Standard	Enables macros that returns formatted output as string.
SparseArrays	Standard	Provides sparse vector/matrice infrastructure and functions.
Statistics	Standard	Offers basic and advanced statistical functions.
CSV	External	Input output interface/infrastructure for comma separated value files. Compatible with DataFrame library.
DataFrames	External	Offers database features with various filters and operations. Data can be processed in the desired format. Supports object-oriented data types.
Plots	External	Offers various plotting functions and features with various back-ends.
StatsPlots	External	Extension of Plots library. Contains many statistical recipes.

3.4. Finite Volume Method Functions

Basic operations regarding finite volume method were coded as functions. Definition of a boundary condition problem is sequential. Problem definition starts with domain description and grid generation. Once computational grid is present at desired resolution, a method for boundary condition inclusion follows. Discretization of the governing equations based on the grid and boundary conditions lead to coefficient matrices of systems of equations. Delivering the coefficient matrices of the problem completes finite volume method sequence of the solution procedure.

3.4.1. FVM Function #1 “initialize_domain”

Generation of the domain is the very first step towards solution. Domain is assumed to be in rectangular shape on two dimensions. Size in each dimension may vary. Grid is assumed to be uniform in x and y directions. Number of divisions may vary in each direction. Thus, “initialize_domain” function works with two-dimensional size input and two number of division input. Table 3.2 summarizes the input variables with brief explanations.

Table 3.2. *Input variables of “initialize_domain” function*

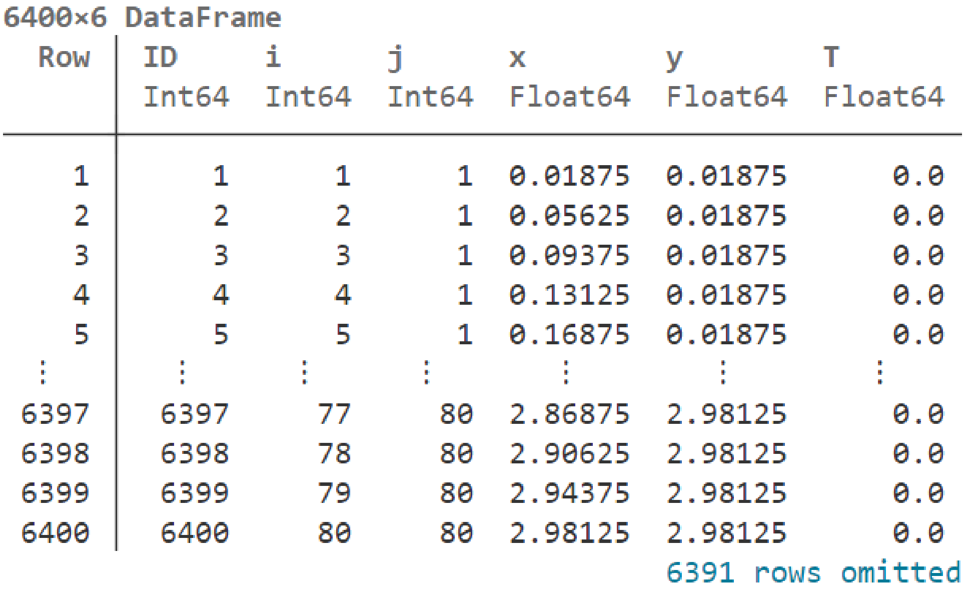
Variable	Type	Explanation
dim1	Float	Length (m) of the domain in first dimension.
dim2	Float	Length (m) of the domain in the second dimension.
n_div_x	Integer	Number of divisions in first (x) direction.
n_div_y	Integer	Number of divisions in second (y) direction.

Number of cells is calculated by multiplying number of divisions on each direction. Size of each uniform cell in both directions are calculated by dividing length of each dimension to number of divisions. DataFrame object is initialized with ID column using number of cell information. Sequentially; columns of indices, coordinates and value (temperature) are calculated and added to DataFrame object simultaneously using DataFrame infrastructure. Property (temperature) values of cells are assigned as zero in order to keep domain dummy for further operations. When all columns are processed, function returns the DataFrame object populated with all individual cell information.

Table 3.3 summarizes the output object while Figure 3.8 is an example output of the “initialize_domain” function. “ID” is the identification number of the cell, “i” is the indice of the cell along the first direction, “j” is the indice of the cell along the second direction, “x” is the coordinate of the cell centre at first direction, “y” is the coordinate of the cell centre at second direction, “T” is the property (temperature) value at cell centre.

Table 3.3. *Output object of “initialize_domain” function*

Object	Type	Explanation
outCs	DataFrame	Container of information (ID, i, j, x, y, T) of grid cells.



```
6400×6 DataFrame
  Row │ ID     i      j      x        y        T
      │ Int64  Int64  Int64  Float64  Float64  Float64
──────┼────────────────────────────────────────────────
    1 │     1      1      1  0.01875  0.01875      0.0
    2 │     2      2      1  0.05625  0.01875      0.0
    3 │     3      3      1  0.09375  0.01875      0.0
    4 │     4      4      1  0.13125  0.01875      0.0
    5 │     5      5      1  0.16875  0.01875      0.0
  ⋮   │   ⋮      ⋮      ⋮       ⋮        ⋮        ⋮
 6397 │  6397     77     80  2.86875  2.98125      0.0
 6398 │  6398     78     80  2.90625  2.98125      0.0
 6399 │  6399     79     80  2.94375  2.98125      0.0
 6400 │  6400     80     80  2.98125  2.98125      0.0
                                     6391 rows omitted
```

Figure 3.8. *Output example of “initialize_domain” function*

3.4.2. FVM Function #2 “boundary_conditions”

Defining boundary conditions is essential for any boundary condition problem. Conventional denotations of boundaries are assumed as west, east, south and north [1]. Fix (“Dirichlet”) and flux (“Neumann”) thermal boundary conditions are assumed. Temperature and heat flux values may vary at each boundary but does not vary within a boundary. Thus, “boundary_conditions” function works with four Dirichlet boundary condition input and four Neumann boundary condition input. Equation 3.3 is given to represent heat flux boundary condition. “k” is the heat transfer coefficient, “A” is the area of the heat transfer, “T” is temperature, “ζ” is the spatial location. Table 3.4 summarizes the input variables with brief explanations.

$$q = kA\frac{dT}{d\zeta} \tag{3.3}$$

Table 3.4. *Input variables of “boundary_conditions” function*

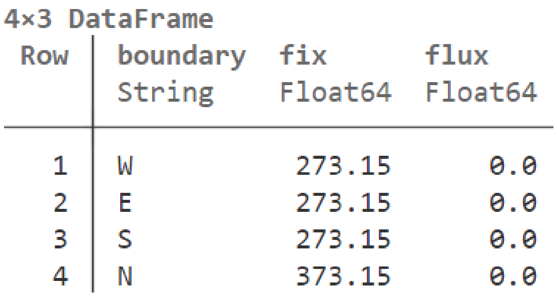
Variable	Type	Explanation
WB_fix	Float	Temperature (K) value of western boundary condition.
WB_flux	Float	Heat flux (W) value of western boundary condition.
EB_fix	Float	Temperature (K) value of eastern boundary condition.
EB_flux	Float	Heat flux (W) value of eastern boundary condition.
SB_fix	Float	Temperature (K) value of southern boundary condition.
SB_flux	Float	Heat flux (W) value of southern boundary condition.
NB_fix	Float	Temperature (K) value of northern boundary condition.
NB_flux	Float	Heat flux (W) value of northern boundary condition.

Empty DataFrame object is created to contain boundary denotations, fix values, flux values. Columns of boundary denotations, temperature values, heat flux values are added into the DataFrame object. Function then returns the DataFrame object contains boundary condition information to be later called as input to other functions.

Table 3.5 summarizes the output object while Figure 3.9 is an example output of the “boundary_conditions” function. “W” denotes west, “E” denotes east, “S” denotes south, “N” denotes north, “fix” is the temperature boundary condition, “flux” is the heat flux boundary condition.

Table 3.5. *Output object of “initialize_domain” function*

Object	Type	Explanation
out_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.



```
4×3 DataFrame
 Row │ boundary  fix      flux
     │ String    Float64  Float64
─────┼───────────────────────────
   1 │ W          273.15      0.0
   2 │ E          273.15      0.0
   3 │ S          273.15      0.0
   4 │ N          373.15      0.0
```

Figure 3.9. *Output example of "boundary_conditions" function*

3.4.3. FVM Function #3 "apply_boundary_conditions"

Validity of solutions based on finite volume method requires proper application of boundary conditions. Additionally, different grid resolutions of multigrid cycles require a compatible method for inclusion of boundary conditions. A method using source terms of finite volume method [1] while preserving discretization equations is generated in order to maintain a viable multigrid cycle execution.

Boundary conditions are assumed to be suitable for the domain towards iterative solution. Sanity check of boundary conditions against divergence is not present. Boundary condition input are assumed as physically feasible. Since reference case is selected, constants and relations of thermal diffusion problem are assumed to be predetermined. Thus, "apply_boundary_conditions" function works with domain and boundary condition input which are assumed to be compatible. Table 3.6 summarizes the input objects with brief explanations.

Table 3.6. *Input objects of "apply_boundary_conditions" function*

Object	Type	Explanation
in_df	DataFrame	Container of information (ID, i, j, x, y, T) of grid cells.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.

Number of divisions and cell sizes in each direction are read/calculated from input DataFrame which contains domain/grid information. Inclusion of boundary conditions requires an imaginary cell addition to perimeter of input grid. Number of divisions in each direction and total cell number is calculated accordingly. DataFrame object is initialized with ID column using number of cell information. Sequentially; columns of indices, coordinates and value (temperature) are calculated and added to DataFrame object simultaneously using DataFrame infrastructure. Temperature values and heat flux of cells are assigned as zero in order to keep domain dummy for further operations. After initialization of the domain with imaginary perimeter addition, boundary conditions values (temperature and heat flux) are assigned to each (W, E, S, N) boundary cells accordingly using DataFrame infrastructure. Source coefficients at boundaries are calculated and assigned to respective cells while simultaneously added as columns to DataFrame object using DataFrame infrastructure. Eventually, a DataFrame object is returned as output containing grid information and boundary conditions in source coefficient format.

Equation 3.4 and Equation 3.5 are used to calculate source coefficients [1]. Figure 3.10 illustrates the transformation of boundary conditions to source terms. Derivation of the source term contributions are given under the "discretised_form" heading below. Source coefficient calculations assume cells to be equal in size at each direction. "k" is the thermal conduction coefficient, "A" is the surface area between cells (by including a constant thickness), "Δζ" is the distance between adjacent cell centres, "q" is the heat flux, "B" subscript denotes the boundary.

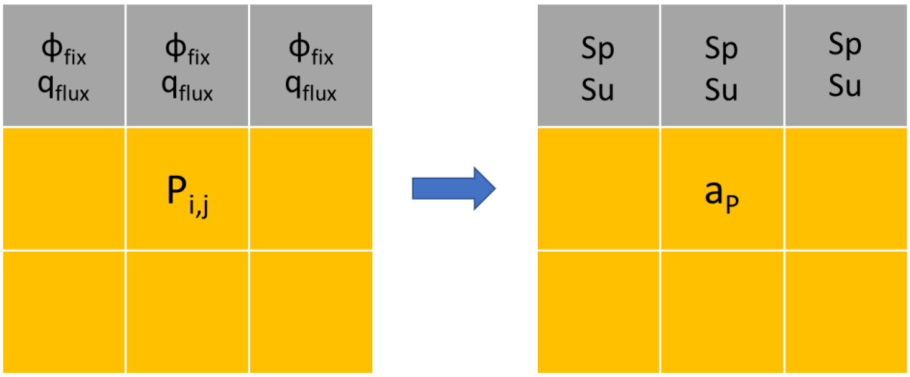


Figure 3.10. *Transformation of boundary conditions to source coefficients*

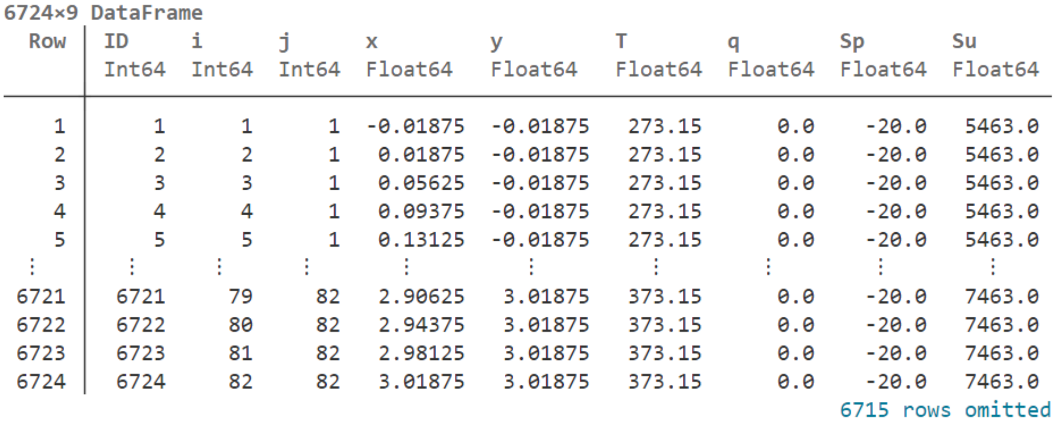
$$S_p = -\frac{2k_B A_B}{\Delta\zeta} \tag{3.4}$$

$$S_u = q_B - S_p\,\phi_B \tag{3.5}$$

Table 3.7 summarizes the output object while Figure 3.11 is an example output of the "apply_boundary_conditions" function. "T" denotes temperature, "q" denotes heat flux, "Sp" and "Su" denotes source coefficients.

Table 3.7. *Output object of "apply_boundary_conditions" function*

Object	Type	Explanation
out_df	DataFrame	Container of grid information (ID, i, j, x, y) and boundary conditions (T, q) in source coefficient (Sp, Su) format.



```
6724×9 DataFrame
 Row │ ID     i      j      x         y         T        q        Sp       Su
     │ Int64  Int64  Int64  Float64   Float64   Float64  Float64  Float64  Float64
─────┼────────────────────────────────────────────────────────────────────────────
   1 │     1      1      1  -0.01875  -0.01875   273.15      0.0    -20.0   5463.0
   2 │     2      2      1   0.01875  -0.01875   273.15      0.0    -20.0   5463.0
   3 │     3      3      1   0.05625  -0.01875   273.15      0.0    -20.0   5463.0
   4 │     4      4      1   0.09375  -0.01875   273.15      0.0    -20.0   5463.0
   5 │     5      5      1   0.13125  -0.01875   273.15      0.0    -20.0   5463.0
  ⋮  │   ⋮      ⋮      ⋮       ⋮         ⋮         ⋮        ⋮        ⋮        ⋮
6721 │  6721     79     82   2.90625   3.01875   373.15      0.0    -20.0   7463.0
6722 │  6722     80     82   2.94375   3.01875   373.15      0.0    -20.0   7463.0
6723 │  6723     81     82   2.98125   3.01875   373.15      0.0    -20.0   7463.0
6724 │  6724     82     82   3.01875   3.01875   373.15      0.0    -20.0   7463.0
                                                                 6715 rows omitted
```

Figure 3.11. *Output example of "apply_boundary_conditions" function*

3.4.4. FVM Function #4 "discretised_form"

Prerequisite to generate coefficient matrices of an iterative solution is to write down system of equations in a discretized form. Discretization method directly effects iterative solution progress alongside grid resolution. Gradients of physical properties are captured either inherently by grid resolution or numerically by discretization method.

Discretization method assumed as central differencing with linear interpolation. Figure 3.12 illustrates cell denotations used in discretization [1] for a two-dimensional finite volume method. "P" denotes the cell being processed, "W" denotes the western cell, "E" denotes eastern cell, "S" denotes southern cell, "N" denotes northern cell, "w" denotes the western cell face, "e" denotes eastern cell face, "s" denotes southern cell face, "n" denotes northern cell face. In short, an uppercase letter denotes cell (centre) while a lowercase letter denotes cell face. Equation 3.6 and Equation 3.7 are used for two-dimensional diffusion problem [1]. Equation 3.8 and Equation 3.9 are the gradually discretized form of the diffusion problem [1]. By distributing Equation 3.7 while using source terms given in Equation 3.10, discretized form is written as seen in Equation 3.11 for two-dimensional diffusion problem [1]. Table 3.8 summarizes the expressions of coefficients of discretized form. "φ" is the physical property, "x" and "y" is the coordinates, “S” denotes source contribution, “V” denotes volume, “Γ” is the relational constant, “A” is the area at interface.

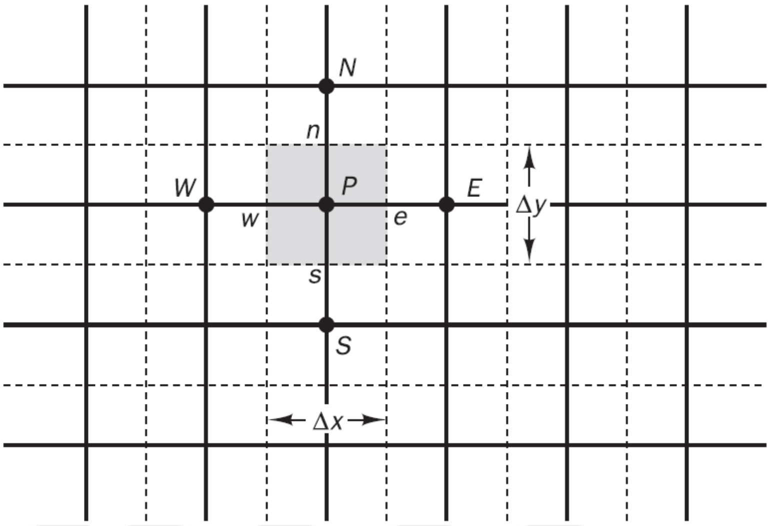


Figure 3.12. *Denotations for discretization function [1]*

$$\frac{\partial}{\partial x}\left(\Gamma\frac{\partial \phi}{\partial \mathrm{x}}\right)+\frac{\partial}{\partial y}\left(\Gamma\frac{\partial \phi}{\partial \mathrm{y}}\right)+S_{\phi}=0 \tag{3.6}$$

$$\int_{\Delta V}\frac{\partial}{\partial x}\left(\Gamma\frac{\partial \phi}{\partial \mathrm{x}}\right)dx\,dy+\int_{\Delta V}\frac{\partial}{\partial y}\left(\Gamma\frac{\partial \phi}{\partial \mathrm{y}}\right)dx\,dy+\int_{\Delta V}S_{\phi}\,dV=0 \tag{3.7}$$

$$\left[\Gamma_e A_e\left(\frac{\partial \phi}{\partial x}\right)_e-\Gamma_w A_w\left(\frac{\partial \phi}{\partial x}\right)_w\right]+\left[\Gamma_s A_s\left(\frac{\partial \phi}{\partial x}\right)_s-\Gamma_n A_n\left(\frac{\partial \phi}{\partial x}\right)_n\right]+\bar{S}\Delta V=0 \tag{3.8}$$

$$\Gamma_e A_e\frac{(\phi_E-\phi_P)}{\delta x_{PE}}-\Gamma_w A_w\frac{(\phi_P-\phi_W)}{\delta x_{WP}}+\Gamma_n A_n\frac{(\phi_N-\phi_P)}{\delta x_{PN}}-\Gamma_s A_s\frac{(\phi_P-\phi_S)}{\delta x_{SP}}+\bar{S}\Delta V=0 \tag{3.9}$$

$$\bar{S}\Delta V=S_u+S_p\phi_P \tag{3.10}$$

Table 3.8. *Coefficients of discretized form*

a_W	a_E	a_S	a_N	a_P
$\frac{\Gamma_w A_w}{\delta x_{WP}}$	$\frac{\Gamma_e A_e}{\delta x_{PE}}$	$\frac{\Gamma_s A_s}{\delta y_{SP}}$	$\frac{\Gamma_n A_n}{\delta y_{PN}}$	$a_W+a_E+a_S+a_N-S_p$

$$a_P\phi_P=a_W\phi_W+a_E\phi_E+a_S\phi_S+a_N\phi_N+S_u \tag{3.11}$$

“discretised_form” function works with DataFrame input which contains domain and boundary condition information. Table 3.9 summarizes the input object with brief explanations.

Table 3.9. *Input object of "discretised form" function*

Object	Type	Explanation
in_df	DataFrame	Container of information (ID, i, j, x, y, T, q, Sp, Su) of grid cells including source terms at perimeter.

Input DataFrame of "discretised_form" function contains additional imaginary cell information at boundaries. Thus, number of divisions of output in each direction and total cell number of output are calculated by excluding imaginary cells at perimeter. DataFrame object is initialized with ID column using number of cell information. Sequentially; columns of indices, coordinates and value (temperature) are calculated and added to DataFrame object simultaneously using DataFrame infrastructure. Dummy columns of coefficients given in Table 3.8 with constant values for aW,aE,aS,aN and zero values for Sp,aP,Su are added to DataFrame object. aW,aE,aS,aN coefficients at boundaries are cleared out by link cutting [1]. Neighbouring (aW,aE,aS,aN) subsets of input DataFrame are taken in order to be used at calculating remaining (Sp,aP,Su) coefficients. Equation 3.12 is used to calculate Sp values, Equation 3.13 is used to calculate aP values and Equation 3.14 is used to calculate Su values. Eventually, a DataFrame object is returned as output containing grid information and respective coefficients of discretization.

$$S_{p_P} = S_{p_W} + S_{p_E} + S_{p_S} + S_{p_N} \tag{3.12}$$

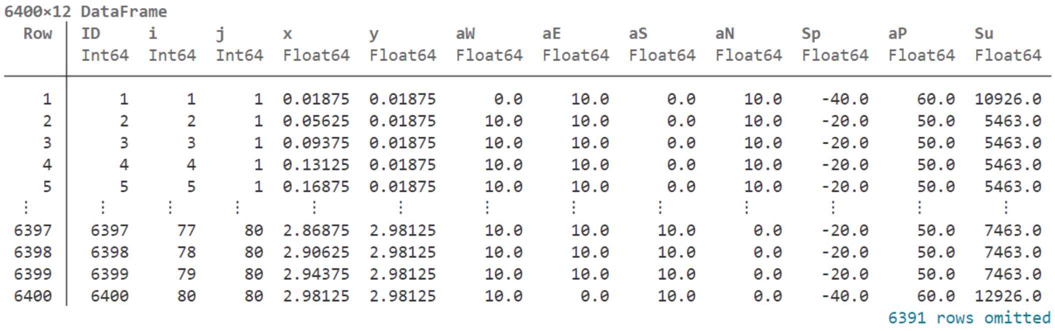
$$a_P = a_W + a_E + a_S + a_N - S_p \tag{3.13}$$

$$S_{u_P} = S_{u_W} + S_{u_E} + S_{u_S} + S_{u_N} \tag{3.14}$$

Table 3.10 summarizes the output object while Figure 3.13 is an example output of the "discretised_form" function.

Table 3.10. *Output object of "apply_boundary_conditions" function*

Object	Type	Explanation
out_df	DataFrame	Container of grid information (ID, i, j, x, y) and (aW, aE, aS, aN, Sp, aP, Su) coefficients of discretization.



```
6400×12 DataFrame
 Row │ ID     i      j      x        y        aW       aE       aS       aN       Sp       aP       Su
     │ Int64  Int64  Int64  Float64  Float64  Float64  Float64  Float64  Float64  Float64  Float64  Float64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      1      1  0.01875  0.01875      0.0     10.0      0.0     10.0    -40.0     60.0  10926.0
   2 │     2      2      1  0.05625  0.01875     10.0     10.0      0.0     10.0    -20.0     50.0   5463.0
   3 │     3      3      1  0.09375  0.01875     10.0     10.0      0.0     10.0    -20.0     50.0   5463.0
   4 │     4      4      1  0.13125  0.01875     10.0     10.0      0.0     10.0    -20.0     50.0   5463.0
   5 │     5      5      1  0.16875  0.01875     10.0     10.0      0.0     10.0    -20.0     50.0   5463.0
  ⋮  │   ⋮      ⋮      ⋮       ⋮        ⋮        ⋮        ⋮        ⋮        ⋮        ⋮        ⋮        ⋮
6397 │  6397     77     80  2.86875  2.98125     10.0     10.0     10.0      0.0    -20.0     50.0   7463.0
6398 │  6398     78     80  2.90625  2.98125     10.0     10.0     10.0      0.0    -20.0     50.0   7463.0
6399 │  6399     79     80  2.94375  2.98125     10.0     10.0     10.0      0.0    -20.0     50.0   7463.0
6400 │  6400     80     80  2.98125  2.98125     10.0      0.0     10.0      0.0    -40.0     60.0  12926.0
                                                                                          6391 rows omitted
```

Figure 3.13. *Output example of "discretised_form" function*

3.4.5. FVM Function #5 "coefficient_matrices"

Last step before venturing towards iterative solution is to generate coefficients of matrices using discretised form of domain. Generating coefficients of matrices is basically writing down system of equations by cell number instead of neighbouring denotations. "coefficient_matrices" function merely transforms/rearranges discretised form of the domain.

"coefficient_matrices" function works with DataFrame input which contains grid information and respective coefficients of discretization. Table 3.11 summarizes the input variables with brief explanations.

Table 3.11. *Input object of "coefficient_matrices" function*

Object	Type	Explanation
in_df	DataFrame	Container of grid information (ID, i, j, x, y) and (aW, aE, aS, aN, Sp, aP, Su) coefficients of discretization.

Number of cells and number of divisions in each direction are read from input DataFrame which contains discretised form of the domain. Dummy coefficient matrices (A) in size of number of cells in two dimension is created for further assignment using zero values. For loop is executed for number of cells, reading neighbouring cell information (aP, aW, aE, aS, aN) and assigning them to corresponding cells according to their IDs at the coefficient matrices. "Su" column of input DataFrame is assigned as constant (b) vector. Eventually, coefficient of matrices (A) and constant vector (b) are returned as an output of "coefficient_matrices" function.

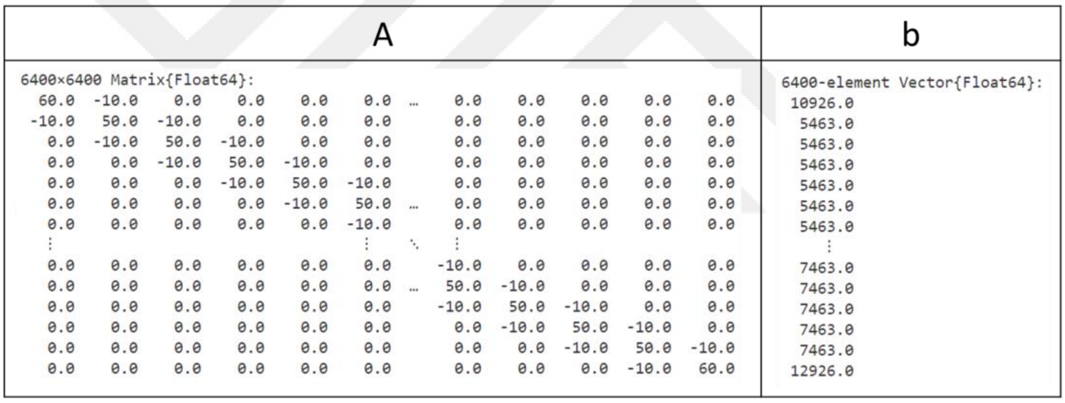
Table 3.12 summarizes the output objects while Figure 3.14 is an example output of the "coefficient_matrices" function. Equation 3.15 and Equation 3.16 show the output format of the matrices. “A” is the coefficient matrices of the system of equations, “x” is the exact solution vector, “b” is the constant vector, “a” are cell coefficients denoted with subscripts of indices, “n” denotes the subscript of indices as well as number of cells, “φ” is the physical property.

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.15}$$

$$A\,x = b \tag{3.16}$$

Table 3.12. *Output objects of “coefficient_matrices” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.



A

```
6400×6400 Matrix{Float64}:
  60.0  -10.0    0.0    0.0    0.0    0.0  …    0.0    0.0    0.0    0.0    0.0
 -10.0   50.0  -10.0    0.0    0.0    0.0       0.0    0.0    0.0    0.0    0.0
   0.0  -10.0   50.0  -10.0    0.0    0.0       0.0    0.0    0.0    0.0    0.0
   0.0    0.0  -10.0   50.0  -10.0    0.0       0.0    0.0    0.0    0.0    0.0
   0.0    0.0    0.0  -10.0   50.0  -10.0       0.0    0.0    0.0    0.0    0.0
   0.0    0.0    0.0    0.0  -10.0   50.0  …    0.0    0.0    0.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0  -10.0       0.0    0.0    0.0    0.0    0.0
   ⋮                                    ⋮    ⋱    ⋮
   0.0    0.0    0.0    0.0    0.0    0.0     -10.0    0.0    0.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0  …   50.0  -10.0    0.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0     -10.0   50.0  -10.0    0.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0       0.0  -10.0   50.0  -10.0    0.0
   0.0    0.0    0.0    0.0    0.0    0.0       0.0    0.0  -10.0   50.0  -10.0
   0.0    0.0    0.0    0.0    0.0    0.0       0.0    0.0    0.0  -10.0   60.0
```

b

```
6400-element Vector{Float64}:
 10926.0
  5463.0
  5463.0
  5463.0
  5463.0
  5463.0
  5463.0
     ⋮
  7463.0
  7463.0
  7463.0
  7463.0
  7463.0
 12926.0
```

Figure 3.14. *Output example of “coefficient_matrices” function*

3.5. Iterative Solution Functions

Various solution methods and algorithms require different iterative solution functions with different conditioning. Thus, solution iterator functions based on iteration count, convergence criteria or special conditionings are generated in order to properly engage in multigrid cycle/algorithm development and exploration. Iterative solution functions are similar in essence with changing conditioning and monitors which serve specific purposes to respective cycles/algorithms.

3.5.1. Iterative Solution Function #1 “gauss_seidel_iterate”

Iterator function is required in order to build iteration functions upon. Gauss-Seidel iterative method without modifications or relaxations is assumed for solution. Equation 3.17 to Equation 3.22 summarizes Gauss-Seidel method used in iterative solutions [3].

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, x = \begin{bmatrix} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_n \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \tag{3.17}$$

$$A = L_* + U \tag{3.18}$$

$$L_* = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad U = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \tag{3.19}$$

$$L_* x = b - Ux \tag{3.20}$$

$$x^{(k+1)} = L_*^{-1}(b - Ux^{(k)}) \tag{3.21}$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)}\right) \quad i = 1,2,\ldots,n \tag{3.22}$$

Iterator function works with matrices (A) and vectors (b, x) input and performs one iteration with Gauss-Seidel method. Table 3.13 summarizes the input objects with brief explanations.

Table 3.13. *Input objects of “gauss_seidel_iterate” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.

Size of the coefficient matrices are read from input. An intermediate operation vector is created in order to contain cumulative row sum of the solution vector. Elements of solution vector are calculated and assigned within the consecutive loops of indices (i, j) which sweeps all rows and non-zero columns. Eventually, solution vector (x) is returned as output of “gauss_seidel_iterate” function. Table 3.14 summarizes the output object.

Table 3.14. *Output object of “gauss_seidel_iterate” function*

Object	Type	Explanation
x	Vector	Container of solution vector.

3.5.2. Iterative Solution Function #2 “iterate_solution_count”

Iteration count is one of the main variables to be monitored through any iterative solution process. Thus, having a function to iterate solution for a certain number of iterations was seemed imperative. “iterate_solution_count” function establishes a base for more advanced iterative solution procedures.

“iterate_solution_count” function works with matrices (A), vectors (b, x) and iteration count (ic) input and performs consecutive iterations. Table 3.15 summarizes the input objects and variables with brief explanations.

Table 3.15. *Input objects of “iterate_solution_count” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
ic	Integer	Number of iterations for the iterative solution.

Upon receiving input, empty/dummy containers (R, S, t, r, y, rh, xh) are created in order to store variables of iterations. “R” is the mean residual vector calculated with Equation 3.24, “S” is the standard deviation of residual vector calculated with Equation 3.25, “t” is the vector of time stamps, “r” is the residual vector calculated with Equation 3.21, “y” is the intermediate solution vector, “rh” is the container of residual vectors. “i” denotes cell number while “n” denotes total number of cells.

$$\vec{r}(i) = |\vec{x}(i) - \vec{y}(i)| \tag{3.23}$$

$$R = mean(\vec{r}) = \frac{1}{n}\sum_1^n \vec{r}(i) \tag{3.24}$$

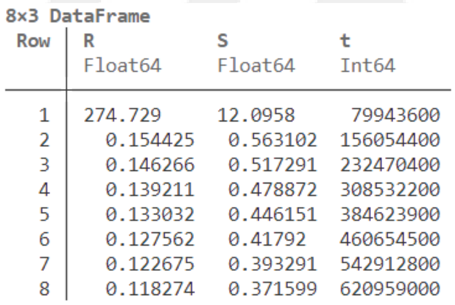
$$S = std(\vec{r}) = \sqrt{\frac{\sum_1^n(\vec{r}(i)-R)^2}{n}} \tag{3.25}$$

Time stamp is used to mark start of the while loop which iterates until the iteration count delivered by input is reached. For each while loop iteration; solution vector is iterated using “gauss_seidel_iterate” function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual vector, current time, residual vector, solution vector) are pushed into respective containers. In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history and residual history objects are returned as output of “iterate_solution_count” function.

Table 3.16 summarizes the output objects while Figure 3.15, Figure 3.16 and Figure 3.17 are example output of the “iterate_solution_count” function.

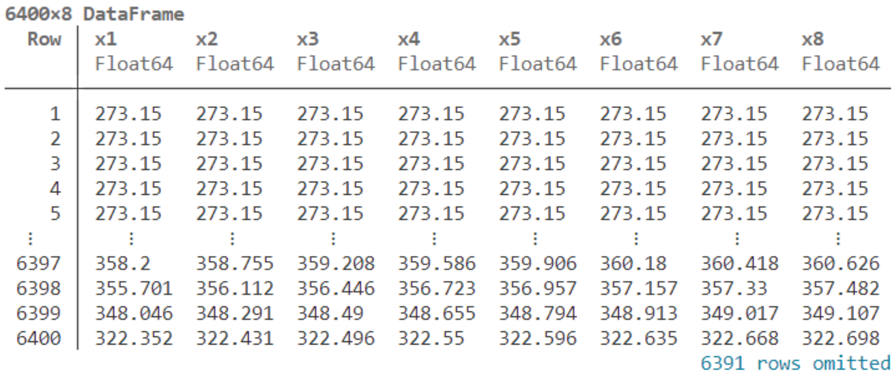
Table 3.16. *Output objects of “iterate_solution_count” function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration



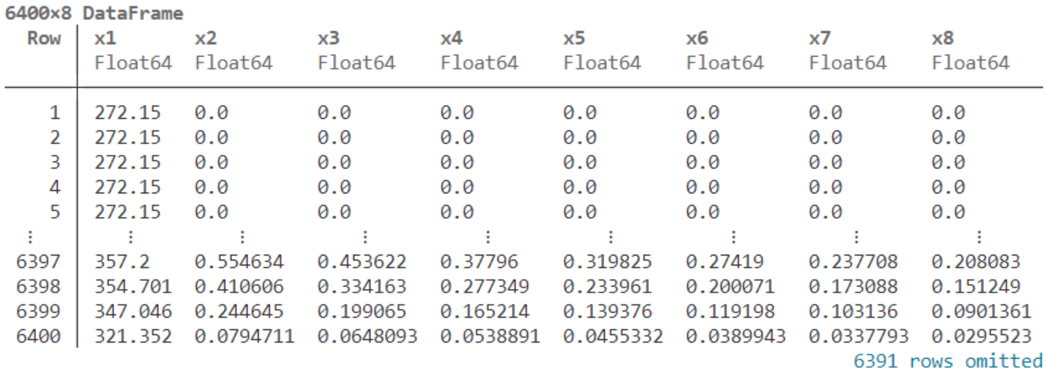
```
8×3 DataFrame
 Row │ R          S          t
     │ Float64    Float64    Int64
─────┼────────────────────────────────
   1 │ 274.729    12.0958     79943600
   2 │   0.154425  0.563102  156054400
   3 │   0.146266  0.517291  232470400
   4 │   0.139211  0.478872  308532200
   5 │   0.133032  0.446151  384623900
   6 │   0.127562  0.41792   460654500
   7 │   0.122675  0.393291  542912800
   8 │   0.118274  0.371599  620959000
```

Figure 3.15. *“iteration_history” output example of “iterate_solution_count” function*



```
6400×8 DataFrame
  Row │ x1       x2       x3       x4       x5       x6       x7       x8
      │ Float64  Float64  Float64  Float64  Float64  Float64  Float64  Float64
──────┼────────────────────────────────────────────────────────────────────────
    1 │ 273.15   273.15   273.15   273.15   273.15   273.15   273.15   273.15
    2 │ 273.15   273.15   273.15   273.15   273.15   273.15   273.15   273.15
    3 │ 273.15   273.15   273.15   273.15   273.15   273.15   273.15   273.15
    4 │ 273.15   273.15   273.15   273.15   273.15   273.15   273.15   273.15
    5 │ 273.15   273.15   273.15   273.15   273.15   273.15   273.15   273.15
  ⋮       ⋮        ⋮        ⋮        ⋮        ⋮        ⋮        ⋮        ⋮
 6397 │ 358.2    358.755  359.208  359.586  359.906  360.18   360.418  360.626
 6398 │ 355.701  356.112  356.446  356.723  356.957  357.157  357.33   357.482
 6399 │ 348.046  348.291  348.49   348.655  348.794  348.913  349.017  349.107
 6400 │ 322.352  322.431  322.496  322.55   322.596  322.635  322.668  322.698
                                                                6391 rows omitted
```

Figure 3.16. *“field_history” output example of “iterate_solution_count” function*



```
6400x8 DataFrame
 Row │ x1       x2         x3         x4         x5         x6         x7         x8
     │ Float64  Float64    Float64    Float64    Float64    Float64    Float64    Float64
─────┼─────────────────────────────────────────────────────────────────────────────────────
   1 │ 272.15   0.0        0.0        0.0        0.0        0.0        0.0        0.0
   2 │ 272.15   0.0        0.0        0.0        0.0        0.0        0.0        0.0
   3 │ 272.15   0.0        0.0        0.0        0.0        0.0        0.0        0.0
   4 │ 272.15   0.0        0.0        0.0        0.0        0.0        0.0        0.0
   5 │ 272.15   0.0        0.0        0.0        0.0        0.0        0.0        0.0
  ⋮  │    ⋮         ⋮          ⋮          ⋮          ⋮          ⋮          ⋮          ⋮
6397 │ 357.2    0.554634   0.453622   0.37796    0.319825   0.27419    0.237708   0.208083
6398 │ 354.701  0.410606   0.334163   0.277349   0.233961   0.200071   0.173088   0.151249
6399 │ 347.046  0.244645   0.199065   0.165214   0.139376   0.119198   0.103136   0.0901361
6400 │ 321.352  0.0794711  0.0648093  0.0538891  0.0455332  0.0389943  0.0337793  0.0295523
                                                                          6391 rows omitted
```

Figure 3.17. *"residual_history" output example of "iterate_solution_count" function*

3.5.3. Iter. Sol. Function #3 "iterate_solution_count_for_cycles"

Iterative solution function for cycles works with matrices (A), vectors (b, x, r) and iteration count (ic) input and performs consecutive iterations. Table 3.17 summarizes the input objects and variables with brief explanations.

Table 3.17. *Input objects of "iterate_solution_count_for_cycles" function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
r	Vector	Container of residual vector.
ic	Integer	Number of iterations for the iterative solution.

Upon receiving input, empty/dummy containers (R, S, t, rh, xh) are created in order to store variables of iterations. Residual vector (r) is read from input and a dummy intermediate solution is calculated with input solution vector (x) and residual vector (r). Time stamp is used to mark start of the while loop which iterates until the iteration count delivered by input is reached. For each while loop iteration; solution vector is iterated using "gauss_seidel_iterate" function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual, current time, residual vector, solution vector) are pushed into respective containers. In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history and residual history objects are returned as output of “iterate_solution_count_for_cycles” function.

Table 3.18 summarizes the output objects of the “iterate_solution_count_for_cycles” function. Example output of “iterate_solution_count_for_cycles” are similar to Figure 3.15, Figure 3.16 and Figure 3.17.

Table 3.18. *Output objects of “iterate_solution_count_for_cycles” function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration

3.5.4. Iter. Sol. Function #4 “iterate_solution_converge_for_cycles”

Iterative solution function for a desired convergence target works with matrices (A), vectors (b, x, r) and a convergence criterion (dc) input and performs consecutive iterations. Table 3.19 summarizes the input objects and variables with brief explanations.

Table 3.19. *Input objects of “iterate_solution_converge_for_cycles” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
r	Vector	Container of residual vector.
dc	Float	Desired convergence level of iterative solution.

Upon receiving input, empty/dummy containers (R, S, t, rh, xh) are created in order to store variables of iterations. Residual vector (r) is read from input and a dummy intermediate solution is calculated with input solution vector (x) and residual vector (r). Time stamp is used to mark start of the while loop which iterates until the convergence criteria delivered by input is reached. While loop is terminated if desired convergence level is reached. For each while loop iteration; solution vector is iterated using “gauss_seidel_iterate” function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual, current time, residual vector, solution vector) are pushed into respective containers. In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history and residual history objects are returned as output of "iterate_solution_converge_for_cycles" function.

Table 3.20 summarizes the output objects of the "iterate_solution_converge_for_cycles" function. Example output of "iterate_solution_converge_for_cycles" are similar to Figure 3.15, Figure 3.16 and Figure 3.17.

Table 3.20. *Output objects of "iterate_solution_converge_for_cycles" function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration

3.5.5. Iter. Sol. Function #5 "iterate_solution_converge_with_count"

Iterative solution function for a certain convergence target works with matrices (A), vectors (b, x), a convergence criterion (cc) and maximum iteration count (i_max) input and performs consecutive iterations. Table 3.21 summarizes the input objects and variables with brief explanations.

Table 3.21. *Input objects of "iterate_solution_converge_with_count" function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
cc	Float	Convergence criteria of iterative solution.
i_max	Integer	Maximum number of iterations for the iterative solution.

Upon receiving input, empty/dummy containers (R, S, t, r, y, rh) are created in order to store variables of iterations. Time stamp is used to mark start of the while loop which iterates until the convergence criteria delivered by input is reached. While loop is terminated if maximum number of iterations is reached. For each while loop iteration; solution vector is iterated using “gauss_seidel_iterate” function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual, current time, residual vector, solution vector) are pushed into respective containers. In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history and residual history objects are returned as output of “iterate_solution_converge_with_count” function.

Table 3.22 summarizes the output objects of the “iterate_solution_converge_with_count” function. Example output of “iterate_solution_converge_with_count” are similar to Figure 3.15, Figure 3.16 and Figure 3.17.

Table 3.22. *Output objects of “iterate_solution_converge_with_count” function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration

3.5.6. Iterative Solution Function #6 “iterate_solution_flexible”

Flexible multigrid cycle requires special iterative solution function which has additional input and output as well as additional conditioning. Even though final convergence target input is essentially same, iterative solution function for flexible cycle also deals with directions in terms of restriction, prolongation or termination. Additional measures for robustness are also incorporated in the process.

“iterate_solution_flexible” function works with matrices (A), vectors (b, x, r), decision criteria (pc, rc), maximum iteration count (i_max), minimum iteration count (i_min) and final convergence target (fc) input and performs consecutive iterations. Table 3.23 summarizes the input variables with brief explanations.

Table 3.23. *Input objects of “iterate_solution_flexible” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
r	Vector	Container of residual vector.
pc	Float	Prolongation criteria.
rc	Float	Restriction criteria.
i_max	Integer	Maximum number of iterations for the iterative solution.
i_min	Integer	Minimum number of iterations during a solution sweep.
fc	Float	Final convergence criterion.

Upon receiving input, empty/dummy containers (R, S, t, rh, xh) are created in order to store variables of iterations. Residual vector (r) is read from input and a dummy intermediate solution is calculated with input solution vector (x) and residual vector (r). Time stamp is used to mark start of the while loop which iterates until any criteria delivered by input is reached. While loop is terminated if maximum number of iterations is reached. For each while loop iteration; solution vector is iterated using “gauss_seidel_iterate” function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual, current time, residual vector, solution vector) are pushed into respective containers.

Restriction criteria, prolongation criteria, termination criteria are continuously checked with each iteration. All criteria trigger respective action registers to be returned as an output towards multigrid cycle that called the “iterative_solution_flexible” function. Table 3.24 summarizes the checks against criteria and respective actions. “cR” is the mean residual of current iteration, “pR” is the mean residual of the previous iteration.

Table 3.24. *Checks against criteria of "iterative_solution_flexible" function and respective actions*

Check	Expression	Action
If restriction criterion is smaller than the ratio of current mean residual to previous mean residual, providing that iteration count is bigger than minimum number of iterations, then…	if $\left(rc < \left(\frac{cR}{pR}\right)\ and\ ic > i_{min}\right)$ then	Restrict
If current mean residual is smaller than the prolongation criterion, providing that iteration count is bigger than minimum number of iterations, then…	if $(cR < pc\ and\ ic > i_{min})$ then	Prolong
If current mean residual is smaller than the final convergence criterion, providing that iteration count is bigger than minimum number of iterations, then…	if $(cR < fc\ and\ ic > i_{min})$ then	Prolong
If iteration count is bigger than maximum number of iterations, then…	if $(\ ic > i_{max})$ then	Prolong

In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history, residual history objects and "action" string are returned as output of "iterate_solution_flexible" function.

Table 3.25 summarizes the output objects of the "iterate_solution_flexible" function. Example output of "iterate_solution_flexible" are similar to Figure 3.15, Figure 3.16 and Figure 3.17 with the addition of "action" string.

Table 3.25. *Output objects of "iterate_solution_flexible" function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration
action	String	Action phrase to be evaluated later.

3.5.7. Iterative Solution Function #7 "iterate_solution_PID_driven"

PID driven multigrid cycle requires special iterative solution function which has additional input and output as well as additional conditioning. Even though final convergence target input is essentially same, iterative solution function for flexible cycle also deals with directions in terms of restriction, prolongation or termination. Additional measures for robustness are also incorporated in the process.

“iterate_solution_PID_driven” function works with matrices (A), vectors (b, x, r), cycle constants (cP, cI, cD), decision criteria (Pc, Ic, Dc), maximum iteration count (i_max), minimum iteration count (i_min) and final convergence target (fc) input and performs consecutive iterations. Table 3.26 summarizes the input variables with brief explanations.

Table 3.26. *Input objects of “iterate_solution_PID_driven” function*

Object	Type	Explanation
A	Matrice	Container of coefficient of matrices based on cell IDs.
b	Vector	Container of constant vector.
x	Vector	Container of solution vector.
r	Vector	Container of residual vector.
cP	Float	Multiplier constant of proportional term.
cI	Float	Multiplier constant of integral term.
cD	Float	Multiplier constant of derivative term.
Pc	Float	Criteria constant to check against proportional term.
Ic	Float	Criteria constant to check against integral term.
Dc	Float	Criteria constant to check against derivative term.
i_max	Integer	Maximum number of iterations for the iterative solution.
i_min	Integer	Minimum number of iterations during a solution sweep.
fc	Float	Final convergence criterion.

Upon receiving input, empty/dummy containers (R, S, t, rh, xh) are created in order to store variables of iterations. Residual vector (r) is read from input and a dummy intermediate solution is calculated with input solution vector (x) and residual vector (r). Dummy values of current mean residual (cR), previous mean residual (pR) and initial mean residual (iR) assigned with mean residual value. Initial mean residual acts as a reference for further condition checks which represents the starting and/or maximum mean residual of the step. Time stamp is used to mark start of the while loop which iterates until any criteria delivered by input is reached. While loop is terminated if maximum number of iterations is reached. For each while loop iteration; solution vector is iterated using “gauss_seidel_iterate” function, residual vector is calculated using current and previous solution vectors, current time stamp is used to record progress, iteration progress variables (mean residual, standard deviation of residual, current time, residual vector, solution vector) are pushed into respective containers. As a robustness precaution, initial mean residual is reassigned as mean residual if it is bigger than initial mean residual within the while loop.

Previous and current error is calculated against final convergence target using previous (pR) and current (cR) residuals respectively. Integral of error is calculated at each iteration using simple trapezoid method with middle point sums. Change of error is calculated with the difference between current and previous error. Proportional, integral and derivative terms are calculated by multiplying coefficients (cP, cI, cD) with current error, integral of error and change of error respectively.

$$previous\ error = pe = pR - fc \tag{3.26}$$

$$current\ error = ce = cR - fc \tag{3.27}$$

$$integral = integral + \left(\frac{ce+pe}{2}\right) \tag{3.28}$$

$$change = ce - pe \tag{3.29}$$

$$proportional\ term = PT = cP * current\ error \tag{3.30}$$

$$integral\ term = IT = cI * integral \tag{3.31}$$

$$derivative\ term = DT = cD * change \tag{3.32}$$

Action assignment criteria (proportional, integral, derivative) are continuously checked with each iteration. All criteria trigger respective action registers to be returned as an output towards multigrid cycle that called the "iterative_solution_PID_driven" function. Table 3.27 summarizes the checks against criteria and respective actions.

Table 3.27. *Checks against criteria of "iterative_solution_PID_driven" function and respective actions*

Check	Expression	Action
If proportional term (PT) is smaller than proportional criteria multiplied by initial mean residual, providing that iteration count is bigger than minimum number of iterations, then…	if $(PT < Pc\ .iR)\ and\ (ic > i_{min})$ then	Prolong
If integral term (IT) is bigger than integral criteria multiplied by initial mean residual, providing that iteration count is bigger than minimum number of iterations, then…	if $(IT > Ic\ .iR)\ and\ (ic > i_{min})$ then	Prolong
If derivative term (DT) is bigger than derivative criteria multiplied by initial mean residual, providing that iteration count is bigger than minimum number of iterations, then…	if $(DT > Dc\ .iR)\ and\ (ic > i_{min})$ then	Restrict
If iteration count is bigger than maximum number of iterations, then…	if $(\ ic > i_{max})$ then	Prolong

In order to keep track of iterative process and later use, history of iterative solution is recorded. Iteration history is recorded by creating a DataFrame object containing mean residual, standard deviation of residual and time stamp columns. Field history is recorded by creating a DataFrame object containing solution vectors of each iteration. Residual history is recorded by creating a DataFrame object containing residual vectors of each iteration. Eventually, iteration history, field history, residual history objects and “action” string are returned as output of “iterate_solution_PID_driven” function.

Table 3.28 summarizes the output objects of the “iterate_solution_PID_driven” function. Example output of “iterate_solution_PID_driven” are similar to Figure 3.15, Figure 3.16 and Figure 3.17 with the addition of “action” string.

Table 3.28. *Output objects of “iterate_solution_PID_driven” function*

Object	Type	Explanation
iteration_history	DataFrame	Container of mean residual vector, standard deviation of residual vector and time stamp vector
field_history	DataFrame	Container of solution vectors of each iteration
residual_history	DataFrame	Container of residual vectors of each iteration
action	String	Action phrase to be evaluated later.

3.6. Intergrid Operation Functions

Initialization, restriction, prolongation and finalization are basic operations for any multigrid cycle. Initialization and finalization operations are covered with iterative solution functions. Restriction and prolongation operations require individual and complimentary functions. Thus, functions that changes the resolution of the solution are generated for multigrid cycles to use.

3.6.1. Intergrid Operation Function #1 “restrict”

Restriction is basically downgrading the resolution. Geometric multigrid operations are assumed. Domain is assumed as rectangular in two dimensions. Grid is assumed cell centred, uniform and structured. Weighting method with linear interpolation is assumed. Equation 3.33 is used to calculate property values of cells at downgraded resolution. Figure 3.18 illustrates restriction operation in two dimensions.

$$C_{i,j}^{h+1} = \frac{1}{4}\left(C_{2i-1,2j-1}^{h} + C_{2i,2j-1}^{h} + C_{2i-1,2j}^{h} + C_{2i,2j}^{h}\right) \tag{3.33}$$

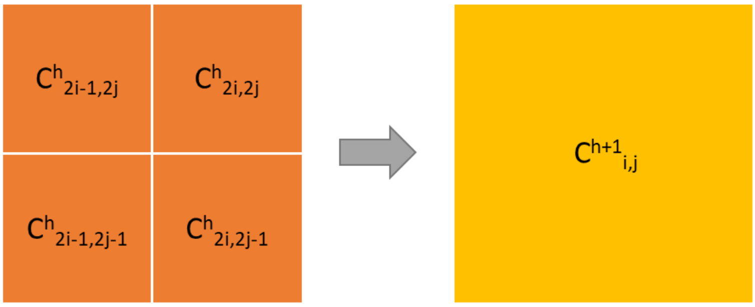


Figure 3.18. *Restriction operation on two dimensions*

Coarsening function works with DataFrame object input that contains grid information (ID, i, j, x, y) and cell properties (T, r). Input is assumed to be compatible for a feasible restriction operation. Table 3.29 summarizes the input objects with brief explanations.

Table 3.29. *Input object of "restriction" function*

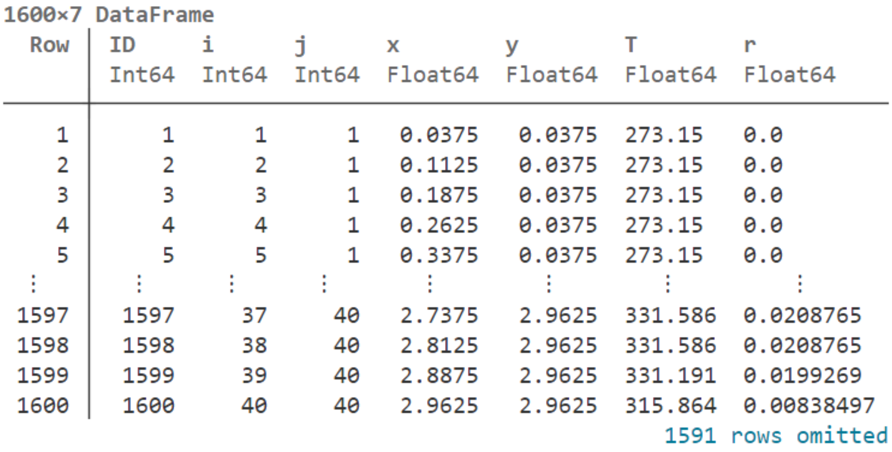
Object	Type	Explanation
in_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r) of fine grid.

Number of divisions of fine grid on each direction are read from input DataFrame. Number of divisions of coarse grid is calculated with number of divisions of fine grid. Number of cells of coarse grid is calculated with number of divisions of coarse grid. Output DataFrame object is initialized with ID column using number of cell information or coarse grid. Columns of indices (i, j) of coarse grid are calculated and added to output DataFrame object simultaneously using DataFrame infrastructure. Dummy columns of cell property values (x, y, T, r) are initialized with zero values. Columns of cell property values (x, y, T, r) of fine grid are reshaped as two-dimensional arrays. Element wise interpolations are carried out on two-dimensional arrays of input cell property matrices to find the coarse grid cell property values. Columns of cell property values (x, y, T, r) of output DataFrame are assigned with the vectorization of interpolated matrices. When all columns are processed, function returns the DataFrame object populated with all individual cell information of the coarse grid.

Table 3.30 summarizes the output object while Figure 3.19 is an example output of the "restrict" function. "ID" is the identification number of the cell, "i" is the indice of the cell along the first direction, "j" is the indice of the cell along the second direction, "x" is the coordinate of the cell centre at first direction, "y" is the coordinate of the cell centre at second direction, "T" is the temperature value at cell centre, "r" is the residual value at the cell centre.

Table 3.30. *Output object of "restrict" function*

Object	Type	Explanation
out_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r) of coarse grid.



```
1600×7 DataFrame
  Row │ ID     i      j      x        y        T        r
      │ Int64  Int64  Int64  Float64  Float64  Float64  Float64
──────┼─────────────────────────────────────────────────────────────
    1 │     1      1      1   0.0375   0.0375  273.15   0.0
    2 │     2      2      1   0.1125   0.0375  273.15   0.0
    3 │     3      3      1   0.1875   0.0375  273.15   0.0
    4 │     4      4      1   0.2625   0.0375  273.15   0.0
    5 │     5      5      1   0.3375   0.0375  273.15   0.0
  ⋮   │   ⋮      ⋮      ⋮       ⋮        ⋮        ⋮         ⋮
 1597 │  1597     37     40   2.7375   2.9625  331.586  0.0208765
 1598 │  1598     38     40   2.8125   2.9625  331.586  0.0208765
 1599 │  1599     39     40   2.8875   2.9625  331.191  0.0199269
 1600 │  1600     40     40   2.9625   2.9625  315.864  0.00838497
                                                 1591 rows omitted
```

Figure 3.19. *Output example of "restriction" function*

3.6.2. Intergrid Operation Function #2 "prolong"

Prolongation is basically upgrading the resolution. Geometric multigrid operations are assumed. Domain is assumed as rectangular in two dimensions. Grid is assumed cell centred, uniform and structured. Linear interpolation is assumed. Equation 3.35 is derived by distributing Equation 3.34 and by arranging the expression. Equations 3.35 to 3.38 are used to calculate property values of cells at upgraded resolution. Figure 3.20 illustrates prolongation operation in two dimensions. Uppercase letters denote coarse grid cells while lowercase letters denote fine grid cells.

$$a = \frac{1}{4}\left(A + \frac{A+B}{2} + \frac{A+C}{2} + \frac{A+B+C+D}{4}\right) \tag{3.34}$$

$$a = \frac{1}{16}(9A + 3B + 3C + D) \tag{3.35}$$

$$b = \frac{1}{16}(3A + 9B + C + 3D) \tag{3.36}$$

$$c = \frac{1}{16}(3A + B + 9C + 3D) \tag{3.37}$$

$$d = \frac{1}{16}(A + 3B + 3C + 9D) \tag{3.38}$$

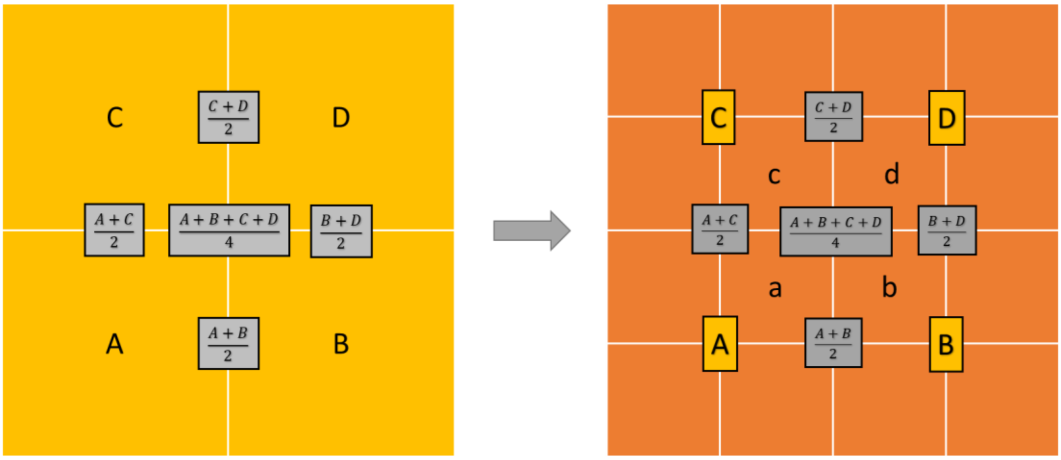


Figure 3.20. *Prolongation operation on two dimensions*

Prolongation function works with boundary condition input in addition to DataFrame object input that contains grid information (ID, i, j, x, y) and cell properties (T, r) while calling "face_and_node_values" function to generate necessary information. Input is assumed to be compatible for a feasible prolongation operation. Table 3.31 summarizes the input objects with brief explanations.

Table 3.31. *Input objects of "prolongation" function*

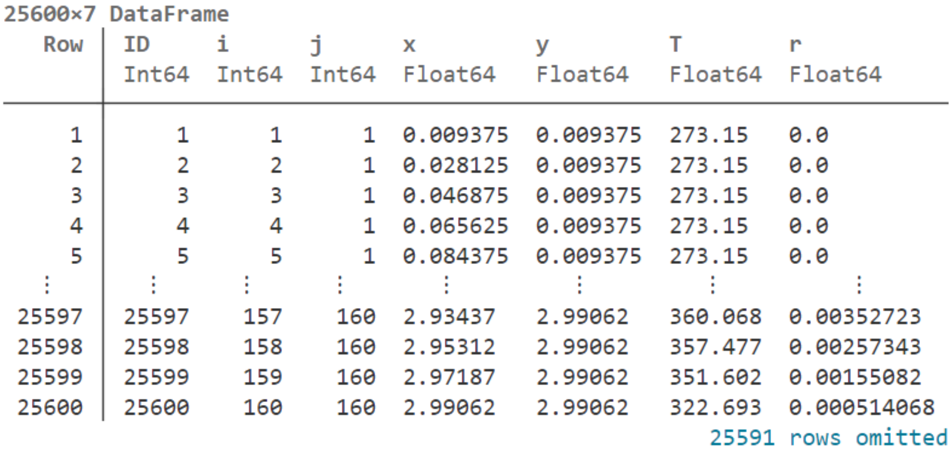
Object	Type	Explanation
in_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r) of coarse grid.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.

Number of divisions of coarse grid on each direction are read from input DataFrame. Number of divisions of fine grid is calculated with number of divisions of fine grid. Number of cells of fine grid is calculated with number of divisions of fine grid. Output DataFrame object is initialized with ID column using number of cell information or fine grid. Columns of indices (i, j) and coordinates (x, y) of fine grid are calculated and added to output DataFrame object simultaneously using DataFrame infrastructure. Temporary DataFrame object is generated to contain the face and node values of each cell of coarse grid by using "face_and_node_values" function in addition to original coarse grid information. Dummy columns of cell property values (T, r) are initialized with zero values. Columns of cell, face and node property values (T, r) of coarse grid are reshaped as two-dimensional arrays. Element wise interpolations are carried out on two-dimensional arrays of input cell, face and node property matrices to find the fine grid cell property values. Columns of cell property values (T, r) of output DataFrame are assigned with the vectorization of interpolated matrices. When all columns are processed, function returns the DataFrame object populated with all individual cell information of the fine grid.

Table 3.32 summarizes the output object while Figure 3.21 is an example output of the “prolong” function. “ID” is the identification number of the cell, “i” is the indice of the cell along the first direction, “j” is the indice of the cell along the second direction, “x” is the coordinate of the cell centre at first direction, “y” is the coordinate of the cell centre at second direction, “T” is the temperature value at cell centre, “r” is the residual value at the cell centre.

Table 3.32. *Output object of “prolong” function*

Object	Type	Explanation
out_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r) of fine grid.



```
25600×7 DataFrame
  Row │ ID     i      j      x         y         T        r
      │ Int64  Int64  Int64  Float64   Float64   Float64  Float64
──────┼─────────────────────────────────────────────────────────────
    1 │     1      1      1  0.009375  0.009375  273.15   0.0
    2 │     2      2      1  0.028125  0.009375  273.15   0.0
    3 │     3      3      1  0.046875  0.009375  273.15   0.0
    4 │     4      4      1  0.065625  0.009375  273.15   0.0
    5 │     5      5      1  0.084375  0.009375  273.15   0.0
  ⋮   │   ⋮      ⋮      ⋮       ⋮         ⋮         ⋮         ⋮
25597 │ 25597    157    160  2.93437   2.99062   360.068  0.00352723
25598 │ 25598    158    160  2.95312   2.99062   357.477  0.00257343
25599 │ 25599    159    160  2.97187   2.99062   351.602  0.00155082
25600 │ 25600    160    160  2.99062   2.99062   322.693  0.000514068
                                                    25591 rows omitted
```

Figure 3.21. *Output example of “prolongation” function*

3.6.3. Intergrid Operation Function #3 “face_and_node_values”

Complication of prolongation operation is the handling of cells at boundaries. Interpolation of the information for the inner cells are straightforward. Generation of a robust prolongation function requires inclusion of boundary condition input and an additional function to generate the information at cell faces and nodes alongside neighbouring cell information.

Figure 3.22 illustrates the denotations for necessary information to be generated with “face_and_node_values” function. Equation 3.39 to 3.46 are used to find property values assuming uniform grid and linear interpolation. “φ” denotes the property value, “P” denotes the cell being processed, “W” denotes the western cell, “E” denotes eastern cell, “S” denotes southern cell, “N” denotes northern cell, “SW” denotes the south western cell, “SE” denotes south eastern cell, “NW” denotes north western cell, “NE” denotes north eastern cell, “w” denotes the western cell face, “e” denotes eastern cell face, “s” denotes southern cell face, “n” denotes northern cell face, “sw” denotes the south western cell node, “se” denotes south eastern cell node, “nw” denotes north western cell node, “ne” denotes north eastern cell node. Uppercase letters denote cells while lowercase letters denote cell faces and cell nodes.

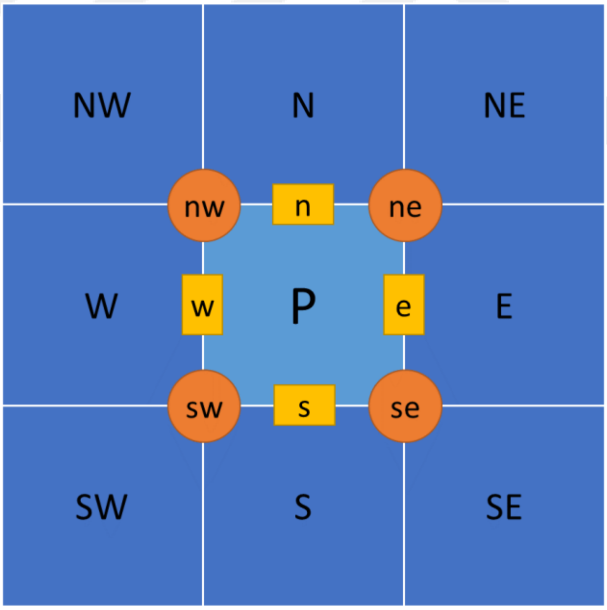


Figure 3.22. *Denotations for “face_and_node_values” function*

$$\phi_{sw} = \frac{1}{4}(\phi_{SW} + \phi_S + \phi_W + \phi_P) \tag{3.39}$$

$$\phi_{se} = \frac{1}{4}(\phi_S + \phi_{SE} + \phi_P + \phi_E) \tag{3.40}$$

$$\phi_{nw} = \frac{1}{4}(\phi_W + \phi_P + \phi_{NW} + \phi_N) \tag{3.41}$$

$$\phi_{ne} = \frac{1}{4}(\phi_P + \phi_E + \phi_N + \phi_{NE}) \tag{3.42}$$

$$\phi_w = \frac{1}{4}(\phi_W + \phi_P + \phi_{sw} + \phi_{nw}) \tag{3.43}$$

$$\phi_e = \frac{1}{4}(\phi_P + \phi_E + \phi_{se} + \phi_{ne}) \tag{3.44}$$

$$\phi_s = \frac{1}{4}(\phi_S + \phi_P + \phi_{sw} + \phi_{se}) \tag{3.45}$$

$$\phi_n = \frac{1}{4}(\phi_P + \phi_N + \phi_{nw} + \phi_{ne}) \tag{3.46}$$

Equation 3.39 to Equation 3.46 does not directly work for cells at boundaries. Special treatment is necessary for cells at each distinct boundary and for each property. Since, temperature and residual are transferred across resolutions via restriction and prolongation, “face_and_node_values” function executes special treatments for temperature and residual values.

Function to generate additional information columns works with boundary condition input and DataFrame object input that contains grid information (ID, i, j, x, y) and cell properties (T, r). Table 3.33 summarizes the input objects with brief explanations.

Table 3.33. *Input objects of “face_and_node_values” function*

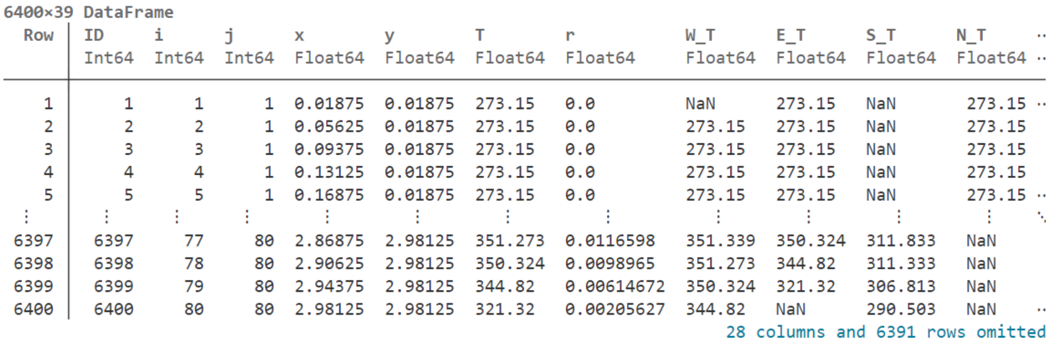
Object	Type	Explanation
in_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r).
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.

Number of divisions in each direction as well as number of cells are read from input DataFrame. Since resolution change is not required, input DataFrame is directly copied as output DataFrame for this stage of the sequence. Residual assumption value at boundaries is assigned. Empty columns are added to output DataFrame for property values of neighbouring cells (W, E, S, N, SW, SE, NW, NE) centres. Empty columns are added to output DataFrame for property values of neighbouring nodes (sw, se, nw, ne) and (w, e, s, n) faces. For loops are executed for number of cells, calculating indices and assigning property values with respective special treatments at corresponding positions. Property values at cell centres that are outside of boundaries are assigned as “NaN”. Property values at faces and nodes that are on boundaries are assigned as respective boundary conditions with proper linear interpolation if necessary. Empty column additions and respective assignment loops are executed for temperature and residual values. Eventually, “face_and_node_values” function returns the DataFrame object populated with additional columns.

Table 3.34 summarizes the output object while Figure 3.23 is an example output of the “face_and_node_values” function.

Table 3.34. *Output object of “face_and_node_values” function*

Object	Type	Explanation
out_Cs	DataFrame	Container of grid information (ID, i, j, x, y) and cell properties (T, r), neighbouring information of cell properties (T, r).



```
6400×39 DataFrame
 Row │ ID     i      j      x        y        T        r           W_T      E_T      S_T      N_T     ⋯
     │ Int64  Int64  Int64  Float64  Float64  Float64  Float64     Float64  Float64  Float64  Float64 ⋯
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      1      1  0.01875  0.01875  273.15   0.0         NaN      273.15   NaN       273.15 ⋯
   2 │     2      2      1  0.05625  0.01875  273.15   0.0         273.15   273.15   NaN       273.15
   3 │     3      3      1  0.09375  0.01875  273.15   0.0         273.15   273.15   NaN       273.15
   4 │     4      4      1  0.13125  0.01875  273.15   0.0         273.15   273.15   NaN       273.15
   5 │     5      5      1  0.16875  0.01875  273.15   0.0         273.15   273.15   NaN       273.15 ⋯
  ⋮  │   ⋮      ⋮      ⋮       ⋮        ⋮        ⋮         ⋮          ⋮        ⋮        ⋮        ⋮    ⋱
6397 │  6397     77     80  2.86875  2.98125  351.273  0.0116598   351.339  350.324  311.833  NaN
6398 │  6398     78     80  2.90625  2.98125  350.324  0.0098965   351.273  344.82   311.333  NaN
6399 │  6399     79     80  2.94375  2.98125  344.82   0.00614672  350.324  321.32   306.813  NaN
6400 │  6400     80     80  2.98125  2.98125  321.32   0.00205627  344.82   NaN      290.503  NaN     ⋯
                                                                      28 columns and 6391 rows omitted
```

Figure 3.23. *Output example of “face_and_node_values” function*

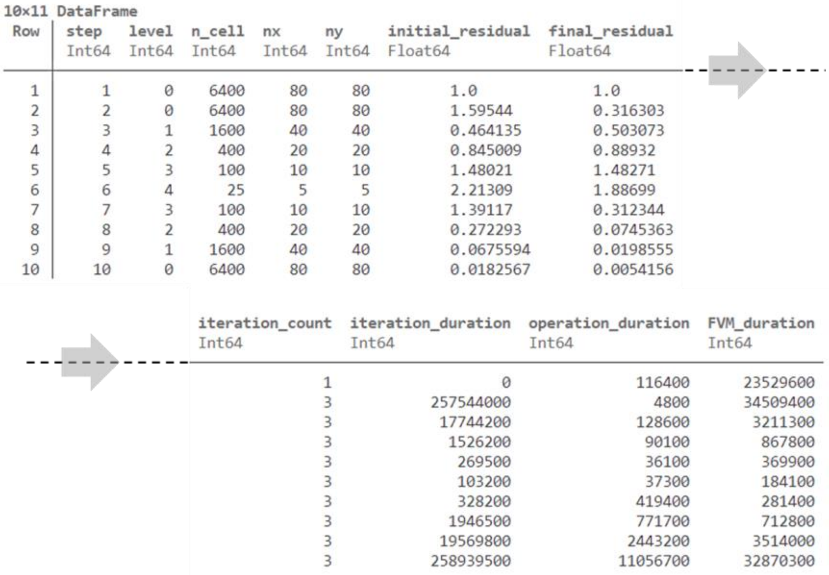
3.7. Multigrid Algorithm Functions

Running an iterative solution with any multigrid cycle requires working algorithms on convenient data structure. Although algorithms may vary, consistent data structure enables fair comparison between cycles/algorithms. Accessibility to recordings of solution progress is another rationale to have a practical data structure. Table 3.35 summarizes the columns of the DataFrame object that records run of a multigrid cycle/algorithm with brief explanations. Some of the columns may be seen as excessive in terms of computational cost, however accessibility and practicality of available information reduced the efforts towards tool development and result generation.

Table 3.35. *Columns of DataFrame object that records the run*

Column	Data Type	Explanation
step	Integer	Steps of the run.
level	Integer	Resolution level of each step.
n_cell	Integer	Number of cells for the resolution on each step.
nx	Integer	Number of divisions in first direction for each step.
ny	Integer	Number of divisions in second direction for each step.
initial_residual	Float	Initial mean residual value at the beginning of each step.
final_residual	Float	Final mean residual value at the end of each step.
iteration_count	Integer	Iteration count for each step.
iteration_duration	Integer	Elapsed time while iterating the solution for each step.
operation_duration	Integer	Elapsed time while executing intergrid (restriction or prolongation) operations preceding finite volume method functions for each step.
FVM_duration	Integer	Elapsed time while executing finite volume method functions preceding iterations for each step.
initial_field	DataFrame	Initial field information (ID, i, j, x, y, T, r) at the beginning of each step.
final_field	DataFrame	Final field information (ID, i, j, x, y, T, r) at the end of each step.
iteration_history	DataFrame	Iterative solution history of mean residual (R), standard deviation of residual (S) and time stamps of each step.
field_history	DataFrame	Solution vector history of iterative solution of each step.
residual_history	DataFrame	Residual vector history of iterative solution of each step.

Excluding columns of DataFrame objects containing field and history information enables summarizing run history in a compact fashion. Figure 3.24 is an example to a run history that summarizes what transpires during a multigrid cycle.



```
10×11 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64
─────┼──────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80         1.0             1.0
   2 │     2      0    6400     80     80         1.59544         0.316303
   3 │     3      1    1600     40     40         0.464135        0.503073
   4 │     4      2     400     20     20         0.845009        0.88932
   5 │     5      3     100     10     10         1.48021         1.48271
   6 │     6      4      25      5      5         2.21309         1.88699
   7 │     7      3     100     10     10         1.39117         0.312344
   8 │     8      2     400     20     20         0.272293        0.0745363
   9 │     9      1    1600     40     40         0.0675594       0.0198555
  10 │    10      0    6400     80     80         0.0182567       0.0054156

                 iteration_count  iteration_duration  operation_duration  FVM_duration
                 Int64            Int64               Int64               Int64
               ────────────────────────────────────────────────────────────────────────
                               1                   0              116400      23529600
                               3           257544000                4800      34509400
                               3            17744200              128600       3211300
                               3             1526200               90100        867800
                               3              269500               36100        369900
                               3              103200               37300        184100
                               3              328200              419400        281400
                               3             1946500              771700        712800
                               3            19569800             2443200       3514000
                               3           258939500            11056700      32870300
```

Figure 3.24. *Example to a run history of a multigrid cycle*

Execution of any multigrid cycle is consisted of calling finite volume method and iterative solution functions while steering through resolution levels either in a predefined or an adaptive fashion. Any multigrid cycle goes through initialization, resolution change (restriction and prolongation), iterative solution sweeps at multigrid levels and finalization activities. Figure 3.25 illustrates conceptual flowchart of any multigrid cycle run. Initialization and finalization activities are at initial resolution level and does not require additional functions rather than iterative solution functions. Pre-process and initial generation of data structure activities are merged under a function. Resolution changes, iteration sweeps, monitoring, conditioning, steering and other complimentary activities of multigrid cycles are merged with initialization and finalization under a function. Post-process of the run is not included within the multigrid algorithms.

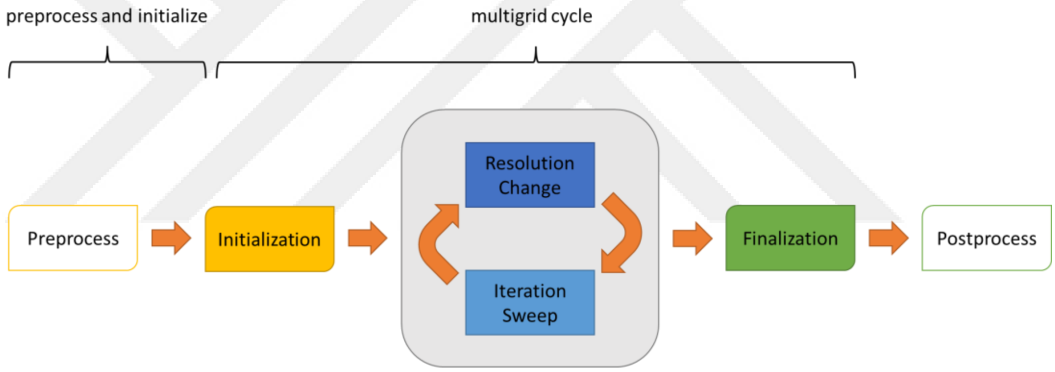


Figure 3.25. *Conceptual flowchart of a multigrid run*

3.7.1. Multigrid Algorithm Function #1 “preprocess_and_initialize”

Every CFD run starts with pre-process activities that generates grid upon domain input while specifying boundary conditions and solver model. Since solver model is predefined with thesis scope, generating grid and executing initial finite volume method operations are assumed to be the pre-process of multigrid cycles. Creating container of run data as described in Table 3.32 and storing initialization phase completes preparation towards execution of multigrid cycles.

“preprocess_and_initialize” function works with domain dimensions and number of division input (dim1, dim2, nx, ny) alongside boundary condition input (bc_df), convergence criteria (cc), iteration count (ic) and initialization value (init_val). Table 3.36 summarizes the input objects with brief explanations.

Table 3.36. *Input variables of "preprocess_and_initialize" function*

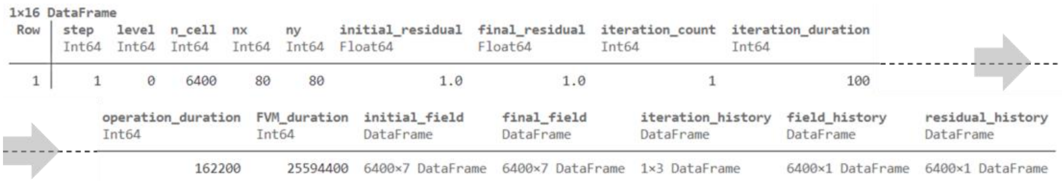
Variable	Type	Explanation
dim1	Float	Length (m) of the domain in first dimension.
dim2	Float	Length (m) of the domain in the second dimension.
nx	Integer	Number of divisions in first (x) direction.
ny	Integer	Number of divisions in second (y) direction.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
cc	Float	Convergence criteria of iterative solution of initialization.
ic	Integer	Number of iterations for the iterative solution of initialization.
init_val	Float	Property (temperature) (K) initialization value for the first iteration.

Number of cells are calculated with number of divisions input. Empty data structure is created as described in Table 3.32 to be populated later. Finite volume method functions are called in sequence using input. Dummy values to residual vector and initialization value to solution vector are assigned during the process. Iterative solution function that monitors convergence and iteration count is called if number of iterations input is bigger than zero to iterate solution. Eventually, data structure is populated with a single row that contains initialization step data and "preprocess_and_initialize" function returns the DataFrame as output.

Table 3.37 summarizes the output object while Figure 3.26 is an example output of the "preprocess_and_initialize" function.

Table 3.37. *Output object of "preprocess_and_initialize" function*

Object	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.35 for initialization phase.



```
1×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80               1.0             1.0                1                 100

      operation_duration  FVM_duration  initial_field       final_field         iteration_history  field_history       residual_history
      Int64               Int64         DataFrame           DataFrame           DataFrame          DataFrame           DataFrame
      ────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
                  162200      25594400  6400×7 DataFrame    6400×7 DataFrame    1×3 DataFrame      6400×1 DataFrame    6400×1 DataFrame
```

Figure 3.26. *Output example of "preprocess_and_initialize" function*

3.7.2. Multigrid Algorithm Function #2 “run_fixed_V_cycle”

Creating algorithms to execute multigrid cycles with predefined patterns are straightforward in terms of complexity. Multigrid cycles with fixed conceptual patterns vary with the iteration count of sweeps on resolution levels in addition to desired or maximum coarsening level. In short, there is not much to steer in terms of algorithm when it comes to multigrid cycles with fixed patterns.

Multigrid cycle that follows fixed V pattern as illustrated in Figure 1.10 belongs to the μ cycle family with parameter value of μ is one [18]. Constant number of iterations per sweep is assumed for resolution levels. Maximum coarsening level is assumed as the maximum feasible level for multigrid methods works. Initialization phase and finalization phase is assumed to be included in the iterations on initial resolution level.

“run_fixed_V_cycle” function works with initialized data structure, boundary conditions, desired coarsening level and number of iteration per sweep input. Table 3.38 summarizes the input objects with brief explanations.

Table 3.38. *Input variables of “run_fixed_V_cycle” function*

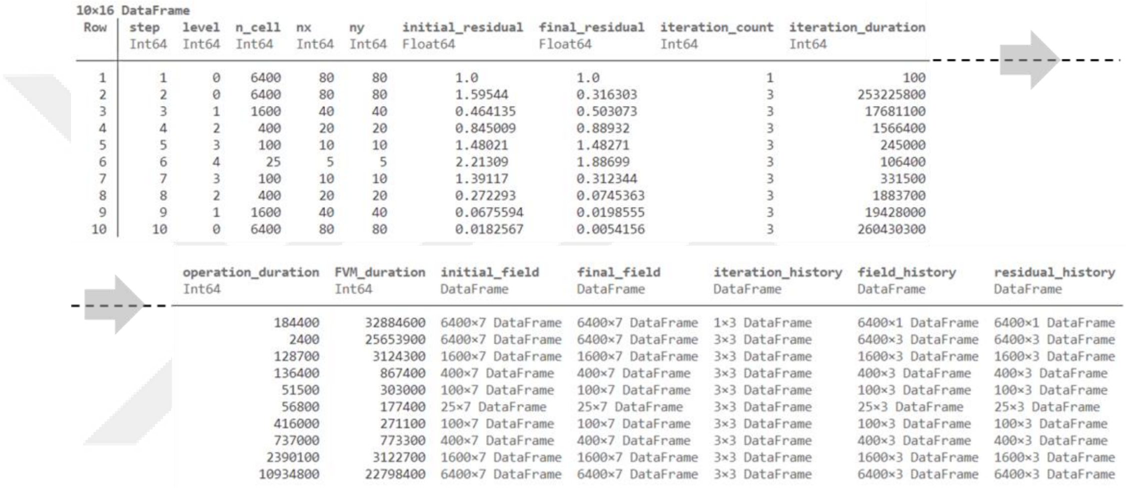
Variable	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.35 for initialization phase.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
course_level	Integer	Desired maximum level of coarsening through the cycle.
i_sweep	Integer	Number of iterations for solution at a resolution.

Initialized data is copied to be later populated with run data. Maximum coarsening level is determined using a complimentary function that checks if desired coarsening level is feasible and assigns maximum possible level instead if it is not. Once maximum coarsening level is set, an array is generated that holds the sequential resolution levels of the cycle. For loop is executed for the level values in the array. Within the for loop; resolution change operations are carried out according to level direction; values of step, number of cells, number of divisions are assigned; finite volume method functions are called for the field on process; iterative solution function for cycles is called according to input; data structure is populated with the information of each step. Finalization phase of multigrid cycle inherently included with the iterative solution of the field on initial resolution at the end of the cycle. Eventually, “run_fixed_V_cycle” function returns DataFrame that contains run data as output.

Table 3.39 summarizes the output object while Figure 3.27 is an example output of the “run_fixed_V_cycle” function.

Table 3.39. *Output object of “run_fixed_V_cycle” function*

Object	Type	Explanation
data	DataFrame	Container of data columns as described in Table 3.35 for multigrid cycle including initialization and finalization.



```
10×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼───────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80         1.0             1.0                       1                 100
   2 │     2      0    6400     80     80         1.59544         0.316303                  3           253225800
   3 │     3      1    1600     40     40         0.464135        0.503073                  3            17681100
   4 │     4      2     400     20     20         0.845009        0.88932                   3             1566400
   5 │     5      3     100     10     10         1.48021         1.48271                   3              245000
   6 │     6      4      25      5      5         2.21309         1.88699                   3              106400
   7 │     7      3     100     10     10         1.39117         0.312344                  3              331500
   8 │     8      2     400     20     20         0.272293        0.0745363                 3             1883700
   9 │     9      1    1600     40     40         0.0675594       0.0198555                 3            19428000
  10 │    10      0    6400     80     80         0.0182567       0.0054156                 3           260430300

 operation_duration  FVM_duration  initial_field       final_field         iteration_history  field_history       residual_history
 Int64               Int64         DataFrame           DataFrame           DataFrame          DataFrame           DataFrame
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
             184400      32884600  6400×7 DataFrame    6400×7 DataFrame    1×3 DataFrame      6400×1 DataFrame    6400×1 DataFrame
               2400      25653900  6400×7 DataFrame    6400×7 DataFrame    3×3 DataFrame      6400×3 DataFrame    6400×3 DataFrame
             128700       3124300  1600×7 DataFrame    1600×7 DataFrame    3×3 DataFrame      1600×3 DataFrame    1600×3 DataFrame
             136400        867400  400×7 DataFrame     400×7 DataFrame     3×3 DataFrame      400×3 DataFrame     400×3 DataFrame
              51500        303000  100×7 DataFrame     100×7 DataFrame     3×3 DataFrame      100×3 DataFrame     100×3 DataFrame
              56800        177400  25×7 DataFrame      25×7 DataFrame      3×3 DataFrame      25×3 DataFrame      25×3 DataFrame
             416000        271100  100×7 DataFrame     100×7 DataFrame     3×3 DataFrame      100×3 DataFrame     100×3 DataFrame
             737000        773300  400×7 DataFrame     400×7 DataFrame     3×3 DataFrame      400×3 DataFrame     400×3 DataFrame
            2390100       3122700  1600×7 DataFrame    1600×7 DataFrame    3×3 DataFrame      1600×3 DataFrame    1600×3 DataFrame
           10934800      22798400  6400×7 DataFrame    6400×7 DataFrame    3×3 DataFrame      6400×3 DataFrame    6400×3 DataFrame
```

Figure 3.27. *Output example of “run_fixed_V_cycle” function*

3.7.3. Multigrid Algorithm Function #3 “run_fixed_W_cycle”

Multigrid cycle that follows fixed W pattern as illustrated in Figure 1.11 belongs to the μ cycle family with parameter value of μ is two [18]. Constant number of iterations per sweep is assumed for resolution levels. Maximum coarsening level is assumed as the maximum feasible level for multigrid methods works. Initialization phase and finalization phase is assumed to be included in the iterations on initial resolution level.

“run_fixed_W_cycle” function works with initialized data structure, boundary conditions, desired coarsening level and number of iteration per sweep input. Table 3.40 summarizes the input objects with brief explanations.

Table 3.40. *Input variables of "run_fixed_W_cycle" function*

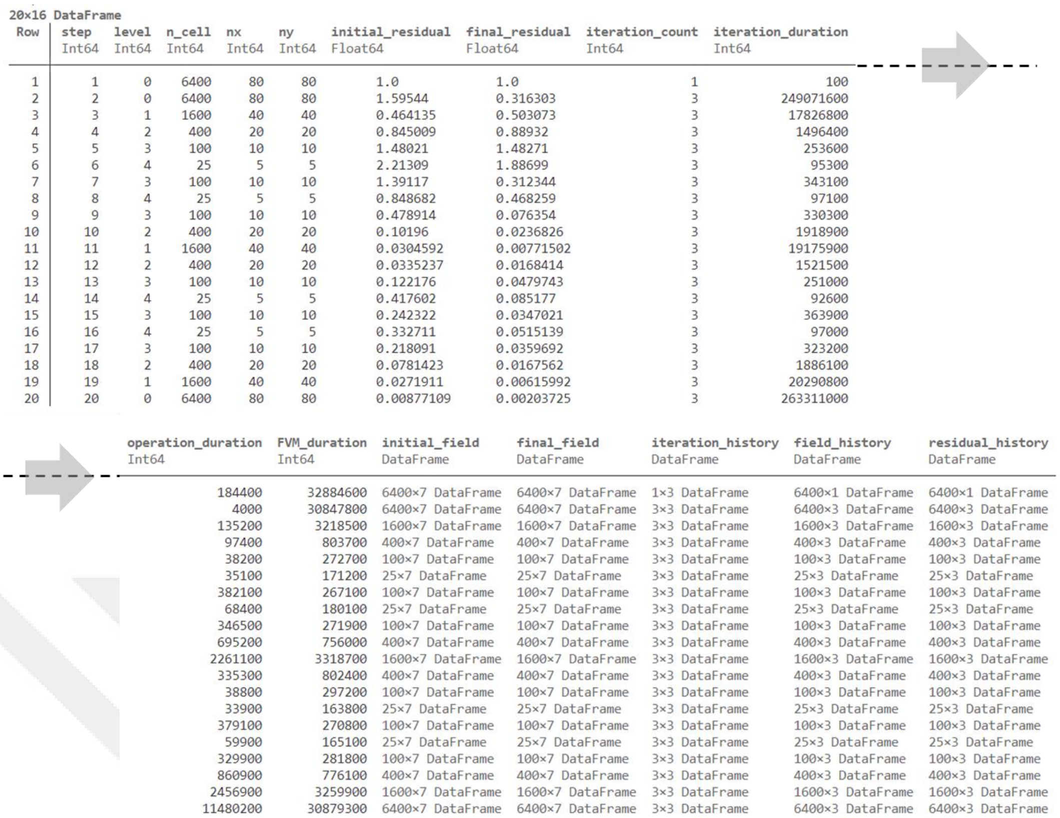
Variable	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.35 for initialization phase.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
course_level	Integer	Desired maximum level of coarsening through the cycle.
i_sweep	Integer	Number of iterations for solution at a resolution.

Initialized data is copied to be later populated with run data. Maximum coarsening level is determined using a complimentary function that checks if desired coarsening level is feasible and assigns maximum possible level instead if it is not. Once maximum coarsening level is set, an array is generated that holds the sequential resolution levels of the cycle. For loop is executed for the level values in the array. Within the for loop; resolution change operations are carried out according to level direction; values of step, number of cells, number of divisions are assigned; finite volume method functions are called for the field on process; iterative solution function for cycles is called according to input; data structure is populated with the information of each step. Finalization phase of multigrid cycle inherently included with the iterative solution of the field on initial resolution at the end of the cycle. Eventually, "run_fixed_W_cycle" function returns DataFrame that contains run data as output.

Table 3.41 summarizes the output object while Figure 3.28 is an example output of the "run_fixed_W_cycle" function.

Table 3.41. *Output object of "run_fixed_W_cycle" function*

Object	Type	Explanation
data	DataFrame	Container of data columns as described in Table 3.35 for multigrid cycle including initialization and finalization.



```
20×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80        1.0               1.0                      1                 100
   2 │     2      0    6400     80     80        1.59544           0.316303                 3           249071600
   3 │     3      1    1600     40     40        0.464135          0.503073                 3            17826800
   4 │     4      2     400     20     20        0.845009          0.88932                  3             1496400
   5 │     5      3     100     10     10        1.48021           1.48271                  3              253600
   6 │     6      4      25      5      5        2.21309           1.88699                  3               95300
   7 │     7      3     100     10     10        1.39117           0.312344                 3              343100
   8 │     8      4      25      5      5        0.848682          0.468259                 3               97100
   9 │     9      3     100     10     10        0.478914          0.076354                 3              330300
  10 │    10      2     400     20     20        0.10196           0.0236826                3             1918900
  11 │    11      1    1600     40     40        0.0304592         0.00771502               3            19175900
  12 │    12      2     400     20     20        0.0335237         0.0168414                3             1521500
  13 │    13      3     100     10     10        0.122176          0.0479743                3              251000
  14 │    14      4      25      5      5        0.417602          0.085177                 3               92600
  15 │    15      3     100     10     10        0.242322          0.0347021                3              363900
  16 │    16      4      25      5      5        0.332711          0.0515139                3               97000
  17 │    17      3     100     10     10        0.218091          0.0359692                3              323200
  18 │    18      2     400     20     20        0.0781423         0.0167562                3             1886100
  19 │    19      1    1600     40     40        0.0271911         0.00615992               3            20290800
  20 │    20      0    6400     80     80        0.00877109        0.00203725               3           263311000

 operation_duration  FVM_duration  initial_field      final_field        iteration_history  field_history      residual_history
 Int64               Int64         DataFrame          DataFrame          DataFrame          DataFrame          DataFrame
─────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
             184400      32884600  6400×7 DataFrame   6400×7 DataFrame   1×3 DataFrame      6400×1 DataFrame   6400×1 DataFrame
               4000      30847800  6400×7 DataFrame   6400×7 DataFrame   3×3 DataFrame      6400×3 DataFrame   6400×3 DataFrame
             135200       3218500  1600×7 DataFrame   1600×7 DataFrame   3×3 DataFrame      1600×3 DataFrame   1600×3 DataFrame
              97400        803700  400×7 DataFrame    400×7 DataFrame    3×3 DataFrame      400×3 DataFrame    400×3 DataFrame
              38200        272700  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
              35100        171200  25×7 DataFrame     25×7 DataFrame     3×3 DataFrame      25×3 DataFrame     25×3 DataFrame
             382100        267100  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
              68400        180100  25×7 DataFrame     25×7 DataFrame     3×3 DataFrame      25×3 DataFrame     25×3 DataFrame
             346500        271900  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
             695200        756000  400×7 DataFrame    400×7 DataFrame    3×3 DataFrame      400×3 DataFrame    400×3 DataFrame
            2261100       3318700  1600×7 DataFrame   1600×7 DataFrame   3×3 DataFrame      1600×3 DataFrame   1600×3 DataFrame
             335300        802400  400×7 DataFrame    400×7 DataFrame    3×3 DataFrame      400×3 DataFrame    400×3 DataFrame
              38800        297200  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
              33900        163800  25×7 DataFrame     25×7 DataFrame     3×3 DataFrame      25×3 DataFrame     25×3 DataFrame
             379100        270800  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
              59900        165100  25×7 DataFrame     25×7 DataFrame     3×3 DataFrame      25×3 DataFrame     25×3 DataFrame
             329900        281800  100×7 DataFrame    100×7 DataFrame    3×3 DataFrame      100×3 DataFrame    100×3 DataFrame
             860900        776100  400×7 DataFrame    400×7 DataFrame    3×3 DataFrame      400×3 DataFrame    400×3 DataFrame
            2456900       3259900  1600×7 DataFrame   1600×7 DataFrame   3×3 DataFrame      1600×3 DataFrame   1600×3 DataFrame
           11480200      30879300  6400×7 DataFrame   6400×7 DataFrame   3×3 DataFrame      6400×3 DataFrame   6400×3 DataFrame
```

Figure 3.28. *Output example of "run_fixed_W_cycle" function*

3.7.4. Multigrid Algorithm Function #4 "run_fixed_F_cycle"

Multigrid cycle that follows fixed F pattern as illustrated in Figure 1.12 [18]. Constant number of iterations per sweep is assumed for resolution levels. Maximum coarsening level is assumed as the maximum feasible level for multigrid methods works. Initialization phase and finalization phase is assumed to be included in the iterations on initial resolution level.

"run_fixed_F_cycle" function works with initialized data structure, boundary conditions, desired coarsening level and number of iteration per sweep input. Table 3.42 summarizes the input objects with brief explanations.

Table 3.42. *Input variables of "run_fixed_F_cycle" function*

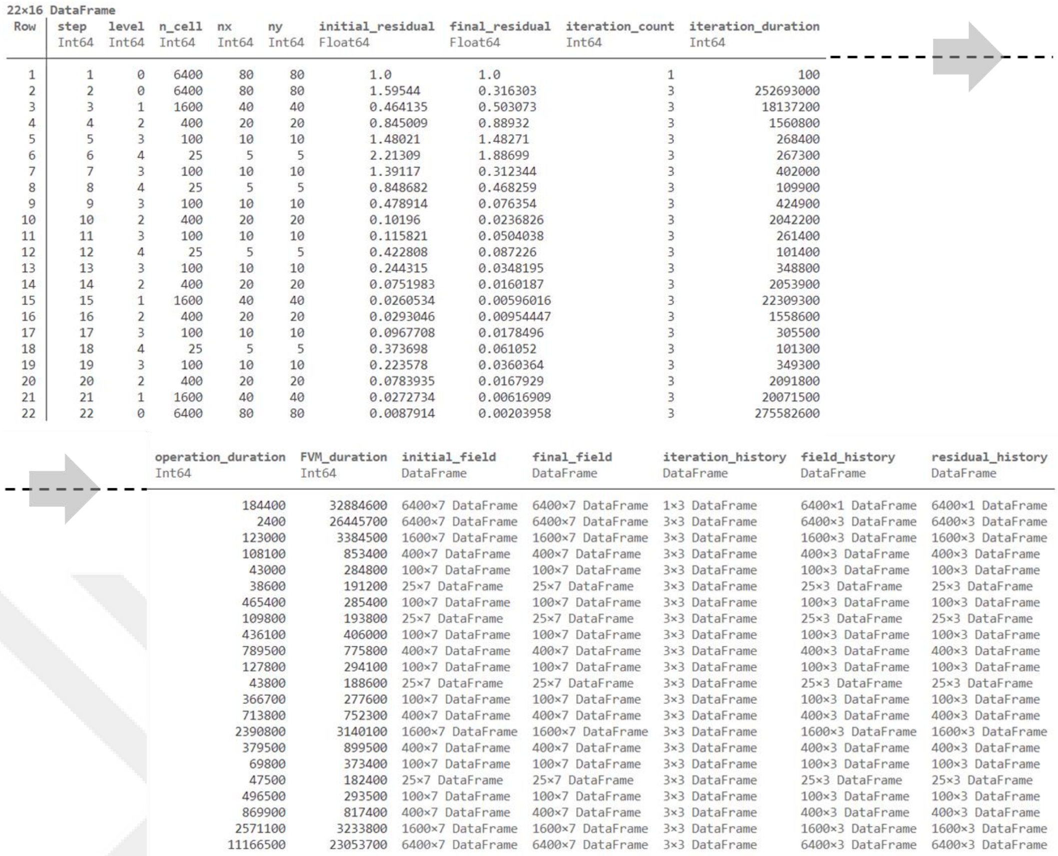
Variable	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.35 for initialization phase.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
course_level	Integer	Desired maximum level of coarsening through the cycle.
i_sweep	Integer	Number of iterations for solution at a resolution.

Initialized data is copied to be later populated with run data. Maximum coarsening level is determined using a complimentary function that checks if desired coarsening level is feasible and assigns maximum possible level instead if it is not. Once maximum coarsening level is set, an array is generated that holds the sequential resolution levels of the cycle. For loop is executed for the level values in the array. Within the for loop; resolution change operations are carried out according to level direction; values of step, number of cells, number of divisions are assigned; finite volume method functions are called for the field on process; iterative solution function for cycles is called according to input; data structure is populated with the information of each step. Finalization phase of multigrid cycle inherently included with the iterative solution of the field on initial resolution at the end of the cycle. Eventually, "run_fixed_F_cycle" function returns DataFrame that contains run data as output.

Table 3.43 summarizes the output object while Figure 3.29 is an example output of the "run_fixed_F_cycle" function.

Table 3.43. *Output object of "run_fixed_F_cycle" function*

Object	Type	Explanation
data	DataFrame	Container of data columns as described in Table 3.35 for multigrid cycle including initialization and finalization.



```
22×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80         1.0              1.0                     1                 100
   2 │     2      0    6400     80     80         1.59544          0.316303                3           252693000
   3 │     3      1    1600     40     40         0.464135         0.503073                3            18137200
   4 │     4      2     400     20     20         0.845009         0.88932                 3             1560800
   5 │     5      3     100     10     10         1.48021          1.48271                 3              268400
   6 │     6      4      25      5      5         2.21309          1.88699                 3              267300
   7 │     7      3     100     10     10         1.39117          0.312344                3              402000
   8 │     8      4      25      5      5         0.848682         0.468259                3              109900
   9 │     9      3     100     10     10         0.478914         0.076354                3              424900
  10 │    10      2     400     20     20         0.10196          0.0236826               3             2042200
  11 │    11      3     100     10     10         0.115821         0.0504038               3              261400
  12 │    12      4      25      5      5         0.422808         0.087226                3              101400
  13 │    13      3     100     10     10         0.244315         0.0348195               3              348800
  14 │    14      2     400     20     20         0.0751983        0.0160187               3             2053900
  15 │    15      1    1600     40     40         0.0260534        0.00596016              3            22309300
  16 │    16      2     400     20     20         0.0293046        0.00954447              3             1558600
  17 │    17      3     100     10     10         0.0967708        0.0178496               3              305500
  18 │    18      4      25      5      5         0.373698         0.061052                3              101300
  19 │    19      3     100     10     10         0.223578         0.0360364               3              349300
  20 │    20      2     400     20     20         0.0783935        0.0167929               3             2091800
  21 │    21      1    1600     40     40         0.0272734        0.00616909              3            20071500
  22 │    22      0    6400     80     80         0.0087914        0.00203958              3           275582600

 operation_duration  FVM_duration  initial_field     final_field       iteration_history  field_history     residual_history
 Int64               Int64         DataFrame         DataFrame         DataFrame          DataFrame         DataFrame
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
             184400      32884600  6400×7 DataFrame  6400×7 DataFrame  1×3 DataFrame      6400×1 DataFrame  6400×1 DataFrame
               2400      26445700  6400×7 DataFrame  6400×7 DataFrame  3×3 DataFrame      6400×3 DataFrame  6400×3 DataFrame
             123000       3384500  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
             108100        853400  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
              43000        284800  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
              38600        191200  25×7 DataFrame    25×7 DataFrame    3×3 DataFrame      25×3 DataFrame    25×3 DataFrame
             465400        285400  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
             109800        193800  25×7 DataFrame    25×7 DataFrame    3×3 DataFrame      25×3 DataFrame    25×3 DataFrame
             436100        406000  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
             789500        775800  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
             127800        294100  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
              43800        188600  25×7 DataFrame    25×7 DataFrame    3×3 DataFrame      25×3 DataFrame    25×3 DataFrame
             366700        277600  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
             713800        752300  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
            2390800       3140100  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
             379500        899500  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
              69800        373400  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
              47500        182400  25×7 DataFrame    25×7 DataFrame    3×3 DataFrame      25×3 DataFrame    25×3 DataFrame
             496500        293500  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
             869900        817400  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
            2571100       3233800  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
           11166500      23053700  6400×7 DataFrame  6400×7 DataFrame  3×3 DataFrame      6400×3 DataFrame  6400×3 DataFrame
```

Figure 3.29. *Output example of "run_fixed_F_cycle" function*

3.7.5. Multigrid Algorithm Function #5 "run_flexible_cycle"

Generating a function executing a flexible multigrid cycle that follows its own path which is determined according to schematic given in Figure 1.14 is no easy task. Number of iterations per sweep may vary for any resolution level. Maximum coarsening level may vary according to conditions. Pattern of the path towards final solution is unknown. Flexible multigrid cycle parameters may vary and are to be explored. Steering of the resolution is assumed to be according to flexible multigrid cycle parameter relation as illustrated in Figure 1.14 while using related equations. Final solution is assumed to be on initial resolution level.

"run_flexible_cycle" function works with initialized data structure, boundary conditions, flexible multigrid cycle parameters (alpha, beta), final convergence criterion, minimum and maximum number of iterations per sweep and maximum number of steps input. Table 3.44 summarizes the input objects with brief explanations.

Table 3.44. *Input variables of "run_flexible_cycle" function*

Variable	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.32 for initialization phase.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
alpha	Float	Flexible cycle parameter related to prolongation condition.
beta	Float	Flexible cycle parameter related to restriction condition.
fc	Float	Convergence criteria of final solution.
i_min	Integer	Maximum number of iterations for solution at a resolution.
i_max	Integer	Maximum number of iterations for solution at a resolution.
step_max	Integer	Maximum number of steps for the multigrid cycle.

Decision dictionary that converts string of action to corresponding integer values as seen in Table 3.42 is created. Initialization data is copied to be later populated with run data. Initial field information is assigned with available initialization data. Action string is set to "continue" and condition variable is set to false. While loop that checks whether condition variable is true or false is used instead of for loop contrary to other fixed cycles. Within the loop, previous level information (step, level, residual) are assigned; current resolution level is calculated using previous level and translation of action; current level information is assigned (step, number of cells, number of divisions); finite volume method functions are called for current field information; solution and residual vectors are assigned according to current field information; prolongation and restriction criteria (pc, rc) are assigned using alpha and beta parameters using Equation 3.47 and Equation 3.48 respectively; prolongation and restriction criteria are reset as described in Table 3.46 if conditions are met as a robustness precaution; iterative solution function for flexible cycles are called with input described in Table 3.23 that returns iteration history, residual history and action string; final field information is copied from initial field information; solution and residual vectors of final field is assigned with output of iterative solution; run data is populated with the information of each step as described in Table 3.35; logic variable regarding restriction feasibility is calculated, projected next level is calculated via current level and decision; condition variable is checked as summarized in Table 3.47; and the while loop is terminated if current step exceeds the maximum number of steps and no other check is triggered. Eventually, "run_flexible_cycle" function returns DataFrame that contains run data as output.

Table 3.45. *Decision dictionary of action strings*

Variable	Type	Translation	Type
restrict	String	1	Integer
prolong	String	-1	Integer
continue	String	0	Integer

$$pc = pR\,.\,alpha \tag{3.47}$$

$$rc = beta \tag{3.48}$$

Table 3.46. *Checks and respective assignments for robustness precautions of flexible cycle*

Check	Expression	Assignment(s)
If prolongation criterion is smaller than final convergence criterion, then…	If $pc < fc$ then	Change prolongation criterion → $pc = fc$
If previous mean residual is smaller than final convergence criterion, then…	If $pR < fc$ then	Change restriction criterion → $\mathrm{rc} = 1$

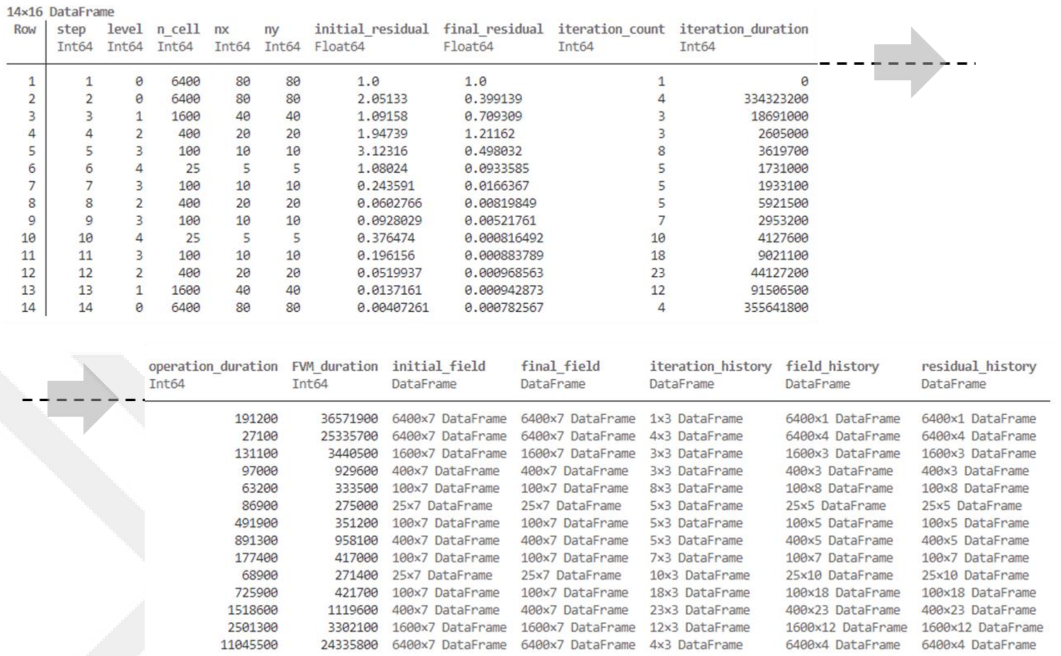
Table 3.47. *Checks and respective assignments for the while loop of flexible cycle*

Check	Expression	Assignment(s) and Action(s)
If level is zero and mean residual is smaller than final convergence criterion, then…	If $(level = 0)$ and $(R < fc)$ then	Terminate loop
If level is zero and projected next level is smaller than the current level, then…	If $(level = 0)$ and $(next\ level < level)$ then	Copy field Change action string to "continue" Continue loop
If level is bigger than zero and mean residual is smaller than final convergence criterion, then…	If $(level > 0)$ and $(R < fc)$ then	Prolong field Change action string to "prolong" Continue loop
If projected next level is bigger than the current level and restriction is infeasible for the available field, then…	If $(next\ level > level)$ and $(restriction\ is\ infeasible)$ then	Copy field Change action string to "continue" Continue loop
If projected next level is bigger than the current level and restriction is feasible for the available field, then…	If $(next\ level > level)$ and $(restriction\ is\ feasible)$ then	Restrict field Continue loop
If level is bigger than zero and projected next level is smaller than the current level, then…	If $(level > 0)$ and $(next\ level < level)$ then	Prolong field Continue loop
If no other check is true, then…	Else	Copy field Set action string Continue loop

Table 3.48 summarizes the output object while Figure 3.30 is an example output of the "run_flexible_cycle" function.

Table 3.48. *Output object of "run_flexible_cycle" function*

Object	Type	Explanation
data	DataFrame	Container of data columns as described in Table 3.35 for multigrid cycle including initialization and finalization.



```
14×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80        1.0           1.0                         1                   0
   2 │     2      0    6400     80     80        2.05133       0.399139                    4           334323200
   3 │     3      1    1600     40     40        1.09158       0.709309                    3            18691000
   4 │     4      2     400     20     20        1.94739       1.21162                     3             2605000
   5 │     5      3     100     10     10        3.12316       0.498032                    8             3619700
   6 │     6      4      25      5      5        1.08024       0.0933585                   5             1731000
   7 │     7      3     100     10     10        0.243591      0.0166367                   5             1933100
   8 │     8      2     400     20     20        0.0602766     0.00819849                  5             5921500
   9 │     9      3     100     10     10        0.0928029     0.00521761                  7             2953200
  10 │    10      4      25      5      5        0.376474      0.000816492                10             4127600
  11 │    11      3     100     10     10        0.196156      0.000883789                18             9021100
  12 │    12      2     400     20     20        0.0519937     0.000968563                23            44127200
  13 │    13      1    1600     40     40        0.0137161     0.000942873                12            91506500
  14 │    14      0    6400     80     80        0.00407261    0.000782567                 4           355641800

 operation_duration  FVM_duration  initial_field     final_field       iteration_history  field_history      residual_history
 Int64               Int64         DataFrame         DataFrame         DataFrame          DataFrame          DataFrame
──────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
             191200      36571900  6400×7 DataFrame  6400×7 DataFrame  1×3 DataFrame      6400×1 DataFrame   6400×1 DataFrame
              27100      25335700  6400×7 DataFrame  6400×7 DataFrame  4×3 DataFrame      6400×4 DataFrame   6400×4 DataFrame
             131100       3440500  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame   1600×3 DataFrame
              97000        929600  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame    400×3 DataFrame
              63200        333500  100×7 DataFrame   100×7 DataFrame   8×3 DataFrame      100×8 DataFrame    100×8 DataFrame
              86900        275000  25×7 DataFrame    25×7 DataFrame    5×3 DataFrame      25×5 DataFrame     25×5 DataFrame
             491900        351200  100×7 DataFrame   100×7 DataFrame   5×3 DataFrame      100×5 DataFrame    100×5 DataFrame
             891300        958100  400×7 DataFrame   400×7 DataFrame   5×3 DataFrame      400×5 DataFrame    400×5 DataFrame
             177400        417000  100×7 DataFrame   100×7 DataFrame   7×3 DataFrame      100×7 DataFrame    100×7 DataFrame
              68900        271400  25×7 DataFrame    25×7 DataFrame    10×3 DataFrame     25×10 DataFrame    25×10 DataFrame
             725900        421700  100×7 DataFrame   100×7 DataFrame   18×3 DataFrame     100×18 DataFrame   100×18 DataFrame
            1518600       1119600  400×7 DataFrame   400×7 DataFrame   23×3 DataFrame     400×23 DataFrame   400×23 DataFrame
            2501300       3302100  1600×7 DataFrame  1600×7 DataFrame  12×3 DataFrame     1600×12 DataFrame  1600×12 DataFrame
           11045500      24335800  6400×7 DataFrame  6400×7 DataFrame  4×3 DataFrame      6400×4 DataFrame   6400×4 DataFrame
```

Figure 3.30. *Output example of "run_flexible_cycle" function*

3.7.6. Multigrid Algorithm Function #6 "run_PID_driven_cycle"

Creating a function executing an adaptive PID driven multigrid cycle that follows its own path which is determined on the way is a challenging task which in fact is also an essential component of the thesis. Number of iterations per sweep may vary for any resolution level. Maximum coarsening level may vary according to conditions. Pattern of the path towards final solution is unknown. PID driven multigrid cycle parameters may vary and are to be explored. Steering of the resolution is assumed to be according to conditional checks between proportional, integral and derivative terms and criteria. Final solution is assumed to be on initial resolution level.

"run_PID_driven_cycle" function works with initialized data structure, boundary conditions, PID coefficients (cP, cI, cD), PID criteria (Pc, Ic, Dc) and operational limitations (fc, i_min, i_max, step_max) input. Table 3.49 summarizes the input objects with brief explanations.

Table 3.49. *Input variables of "run_PID_driven_cycle" function*

Variable	Type	Explanation
init_data	DataFrame	Container of data columns as described in Table 3.32 for initialization phase.
bc_df	DataFrame	Container of boundary condition information at domain boundaries in terms of temperature and heat flux.
cP	Float	Multiplier constant of proportional term.
cI	Float	Multiplier constant of integral term.
cD	Float	Multiplier constant of derivative term.
Pc	Float	Criteria constant to check against proportional term.
Ic	Float	Criteria constant to check against integral term.
Dc	Float	Criteria constant to check against derivative term.
fc	Float	Convergence criteria of final solution.
i_min	Integer	Maximum number of iterations for solution at a resolution.
i_max	Integer	Maximum number of iterations for solution at a resolution.
step_max	Integer	Maximum number of steps for the multigrid cycle.

Decision dictionary that converts string of action to corresponding integer values as seen in Table 3.50 is created. Input parameters are assigned to corresponding individual variables for further use. Initialization data is copied to be later populated with run data. Initial field information is assigned with available initialization data. Action string is set to "continue" and progress variable is set to true. While loop that checks whether progress variable is true or false is used instead of for loop contrary to other fixed cycles. Within the loop, previous level information (step, level, residual) are assigned; current resolution level is calculated using previous level and translation of action; current level information is assigned (step, number of cells, number of divisions); logic variable regarding restriction feasibility is calculated; finite volume method functions are called for current field information; solution and residual vectors are assigned according to current field information; proportional, integral and derivative criteria assigned using PID criteria input parameters; proportional, integral and derivative criteria are reset as described in Table 3.51 if conditions are met as a robustness precaution; iterative solution function for PID driven cycles are called with input described in Table 3.26 that returns iteration history, residual history and action string; final field information is copied from initial field information; solution and residual vectors of final field is assigned with output of iterative solution; run data is populated with the information of each step as described in Table 3.35; projected next level is calculated via current level and decision; progress variable is checked as summarized in Table 3.52; and the while loop is terminated if current step exceeds the maximum number of steps and no other check is triggered.

Eventually, “run_PID_driven_cycle” function returns DataFrame that contains run data as output.

Table 3.50. *Decision dictionary of action strings*

Variable	Type	Translation	Type
restrict	String	1	Integer
prolong	String	-1	Integer
continue	String	0	Integer

Table 3.51. *Checks and respective assignments for robustness precautions of PID driven cycle*

Check	Expression	Assignment(s)
If level is smaller than one, then…	If $(level < 1)$ then	Change proportional criterion → $Pc = 0$
If restriction is infeasible, then…	If $(restriction\ infesible)$ then	Change integral criterion → $Ic = \frac{1}{fc}$
If previous mean residual is smaller than final convergence criterion, then…	If $(pR < fc)$ then	Change derivative criterion → $Dc = 0$

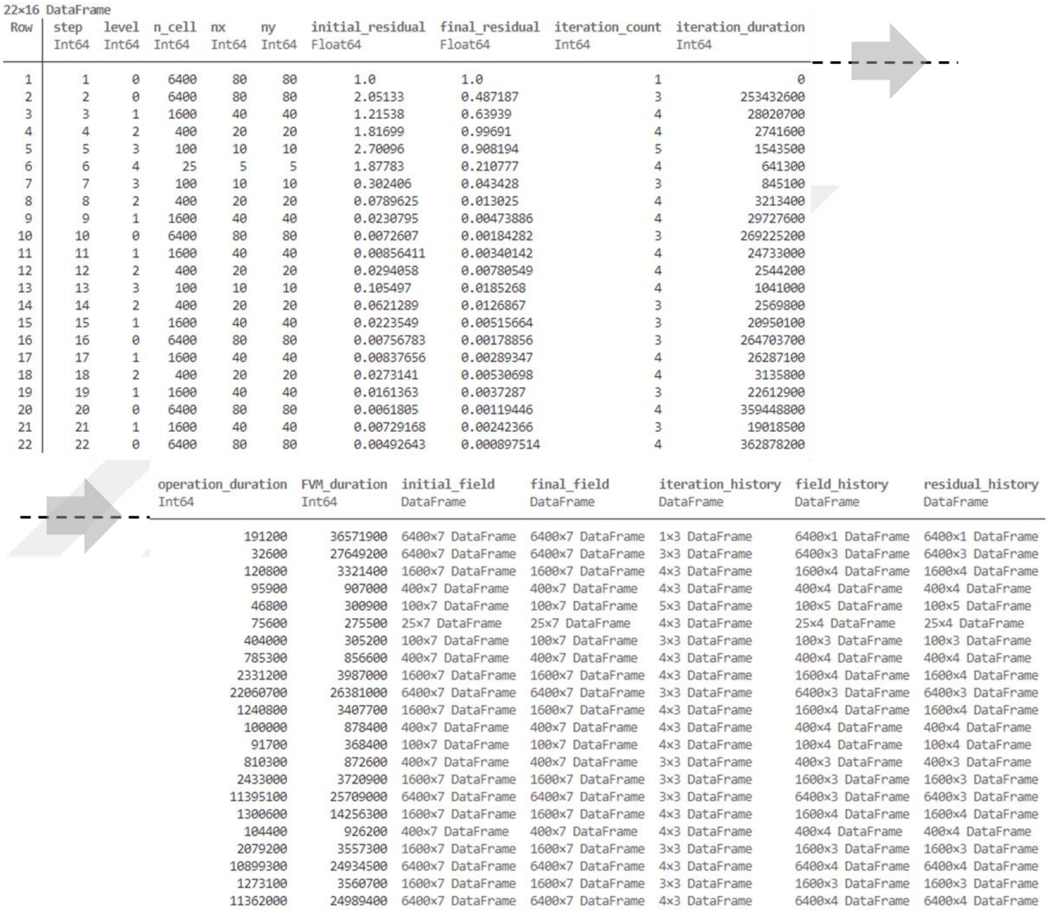
Table 3.52. *Checks and respective assignments for the while loop of PID driven cycle*

Check	Expression	Assignment(s) and Action(s)
If level is zero and mean residual is smaller than final convergence criterion, then…	If $(level = 0)\ and\ (R < fc)$ then	Terminate loop
If level is zero and projected next level is smaller than the current level, then…	If $(level = 0)\ and$ $(next\ level < level)$ then	Copy field Change action string to “continue” Continue loop
If level is bigger than zero and mean residual is smaller than final convergence criterion, then…	If $(level > 0)\ and\ (R < fc)$ then	Prolong field Change action string to “prolong” Continue loop
If projected next level is bigger than the current level and restriction is infeasible for the available field, then…	If $(next\ level > level)\ and$ $(restriction\ is\ infeasible)$ then	Copy field Change action string to “continue” Continue loop
If projected next level is bigger than the current level and restriction is feasible for the available field, then…	If $(next\ level > level)\ and$ $(restriction\ is\ feasible)$ then	Restrict field Continue loop
If level is bigger than zero and projected next level is smaller than the current level, then…	If $(level > 0)\ and$ $(next\ level < level)$ then	Prolong field Continue loop
If no other check is true, then…	Else	Copy field Set action string Continue loop

Table 3.53 summarizes the output object while Figure 3.31 is an example output of the “run_PID_driven_cycle” function.

Table 3.53. *Output object of “run_PID_driven_cycle” function*

Object	Type	Explanation
data	DataFrame	Container of data columns as described in Table 3.35 for multigrid cycle including initialization and finalization.



```
22×16 DataFrame
 Row │ step   level  n_cell  nx     ny     initial_residual  final_residual  iteration_count  iteration_duration
     │ Int64  Int64  Int64   Int64  Int64  Float64           Float64         Int64            Int64
─────┼──────────────────────────────────────────────────────────────────────────────────────────────────────────
   1 │     1      0    6400     80     80        1.0            1.0                        1                   0
   2 │     2      0    6400     80     80        2.05133        0.487187                   3           253432600
   3 │     3      1    1600     40     40        1.21538        0.63939                    4            28020700
   4 │     4      2     400     20     20        1.81699        0.99691                    4             2741600
   5 │     5      3     100     10     10        2.70096        0.908194                   5             1543500
   6 │     6      4      25      5      5        1.87783        0.210777                   4              641300
   7 │     7      3     100     10     10        0.302406       0.043428                   3              845100
   8 │     8      2     400     20     20        0.0789625      0.013025                   4             3213400
   9 │     9      1    1600     40     40        0.0230795      0.00473886                 4            29727600
  10 │    10      0    6400     80     80        0.0072607      0.00184282                 3           269225200
  11 │    11      1    1600     40     40        0.00856411     0.00340142                 4            24733000
  12 │    12      2     400     20     20        0.0294058      0.00780549                 4             2544200
  13 │    13      3     100     10     10        0.105497       0.0185268                  4             1041000
  14 │    14      2     400     20     20        0.0621289      0.0126867                  3             2569800
  15 │    15      1    1600     40     40        0.0223549      0.00515664                 3            20950100
  16 │    16      0    6400     80     80        0.00756783     0.00178856                 3           264703700
  17 │    17      1    1600     40     40        0.00837656     0.00289347                 4            26287100
  18 │    18      2     400     20     20        0.0273141      0.00530698                 4             3135800
  19 │    19      1    1600     40     40        0.0161363      0.0037287                  3            22612900
  20 │    20      0    6400     80     80        0.0061805      0.00119446                 4           359448800
  21 │    21      1    1600     40     40        0.00729168     0.00242366                 3            19018500
  22 │    22      0    6400     80     80        0.00492643     0.000897514                4           362878200

 operation_duration  FVM_duration  initial_field     final_field       iteration_history  field_history     residual_history
 Int64               Int64         DataFrame         DataFrame         DataFrame          DataFrame         DataFrame
────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────────
             191200      36571900  6400×7 DataFrame  6400×7 DataFrame  1×3 DataFrame      6400×1 DataFrame  6400×1 DataFrame
              32600      27649200  6400×7 DataFrame  6400×7 DataFrame  3×3 DataFrame      6400×3 DataFrame  6400×3 DataFrame
             120800       3321400  1600×7 DataFrame  1600×7 DataFrame  4×3 DataFrame      1600×4 DataFrame  1600×4 DataFrame
              95900        907000  400×7 DataFrame   400×7 DataFrame   4×3 DataFrame      400×4 DataFrame   400×4 DataFrame
              46800        300900  100×7 DataFrame   100×7 DataFrame   5×3 DataFrame      100×5 DataFrame   100×5 DataFrame
              75600        275500  25×7 DataFrame    25×7 DataFrame    4×3 DataFrame      25×4 DataFrame    25×4 DataFrame
             404000        305200  100×7 DataFrame   100×7 DataFrame   3×3 DataFrame      100×3 DataFrame   100×3 DataFrame
             785300        856600  400×7 DataFrame   400×7 DataFrame   4×3 DataFrame      400×4 DataFrame   400×4 DataFrame
            2331200       3987000  1600×7 DataFrame  1600×7 DataFrame  4×3 DataFrame      1600×4 DataFrame  1600×4 DataFrame
           22060700      26381000  6400×7 DataFrame  6400×7 DataFrame  3×3 DataFrame      6400×3 DataFrame  6400×3 DataFrame
            1240800       3407700  1600×7 DataFrame  1600×7 DataFrame  4×3 DataFrame      1600×4 DataFrame  1600×4 DataFrame
             100000        878400  400×7 DataFrame   400×7 DataFrame   4×3 DataFrame      400×4 DataFrame   400×4 DataFrame
              91700        368400  100×7 DataFrame   100×7 DataFrame   4×3 DataFrame      100×4 DataFrame   100×4 DataFrame
             810300        872600  400×7 DataFrame   400×7 DataFrame   3×3 DataFrame      400×3 DataFrame   400×3 DataFrame
            2433000       3720900  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
           11395100      25709000  6400×7 DataFrame  6400×7 DataFrame  3×3 DataFrame      6400×3 DataFrame  6400×3 DataFrame
            1300600      14256300  1600×7 DataFrame  1600×7 DataFrame  4×3 DataFrame      1600×4 DataFrame  1600×4 DataFrame
             104400        926200  400×7 DataFrame   400×7 DataFrame   4×3 DataFrame      400×4 DataFrame   400×4 DataFrame
            2079200       3557300  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
           10899300      24934500  6400×7 DataFrame  6400×7 DataFrame  4×3 DataFrame      6400×4 DataFrame  6400×4 DataFrame
            1273100       3560700  1600×7 DataFrame  1600×7 DataFrame  3×3 DataFrame      1600×3 DataFrame  1600×3 DataFrame
           11362000      24989400  6400×7 DataFrame  6400×7 DataFrame  4×3 DataFrame      6400×4 DataFrame  6400×4 DataFrame
```

Figure 3.31. *Output example of “run_PID_driven_cycle” function*

3.8. Complimentary Functions

Even though core functions are given thoroughly; there are complimentary functions which are called in main functions or individually which are not explained in detail. Code of complimentary functions are not given in appendices. Auxiliary functions are mainly basic tools carrying out simple but necessary tasks. Visualization functions are mainly postprocess tools which illustrates given data. Exploration functions are mainly numerical experimentation setups that sweeps through a parameter space for a specific run type. Result Generation functions are for generating results packs with predetermined context; in order to compile data and illustrations which provides consistent and comparable output.

3.8.1. Auxiliary Functions

“generate_laplace_solution” function creates a solution domain similar to “initialize_domain” but instead of creating a dummy solution vector, it fills the solution vector with exact solution. This function calls another auxiliary function called “reference_case_temperature” which provides calculation of exact solution at input location.

“measure_reference_work_unit” function finds the reference work unit of given run data.

“filter_time_per_iteration” function filters outliers of given time per iteration values using standart deviation.

“find_max_level” function finds the maximum feasible coarse level of given grid.

“calculate_cost_ratios” function calculates the cost ratios of FVM functions, intergrid operations, iterations and total cycle.

“construct_array_of_vectors” function creates an array of vectors with given ranges of parameters. This function is called prior to exploration functions and is used to provide exploration space input.

“take_diagonal_subset” function generates a solution vector for further postprocessing by taking the diagonal subset of the given field.

“add_data_for_comparison_plot” function creates a container that holds rows of data and properties. This container enables illustrating comparison plots within loops in an orderly fashion.

3.8.2. Visualization Functions

“plot_residual_scatter” function generates the illustration of grid cells with colouring according to residual value for given step and iteration of given run data. “plot_residual_scatter_with_field_data” function is a variant which accepts field data as input.

"plot_field_contour" function generates the illustration of coloured field contour of the solution property for given step and iteration of given run data. "plot_field_contour_with_field_data" function is a variant which accepts field data as input.

"plot_residual_progress" function generates the plot of residual progress of given run data with marking initial and final residual values of each step.

"plot_time_per_iteration" function generates the plot of elapsed time values with each iteration of given direct iterative run data. This function calls "filter_time_per_iteration" in order to present both filtered and unfiltered graph of elapsed time per iteration values.

"plot_run_summary" function generates the illustration of comprehensive yet minimalistic presentation of the given run data in a summarized form. This function generates various annotations to emphasize main variables of the run. This function calls "calculate_cost_ratios" function to get some of data required for the annotations. This function returns summary metrics array which is comprised of reference work unit and cost ratios; in addition to the actual run summary plot.

"plot_surface" function is a generic tool for create three dimensional surface plots. This function accepts x, y and z values to plot and colours the surface output according to z values. This function is mainly called while generating exploration visualizations.

"plot_stats_box" and "plot_stats_density" functions generate the comparison illustration of given data in the form of statistical box plot and probability density plot respectively.

"plot_diagonal_values" function generate comparison plot of diagonal values of given data.

3.8.3. Exploration Functions

"explore_resolutions_of_direct_runs" carries out numerical experiments regarding grid resolutions of direct iterative solution runs. This function explores the effect of varying grid resolution around a given reference case. This function records values of number of cells, iteration count and reference work units of each direct iterative solution run with varying grid resolution.

“explore_boundary_conditions_of_direct_runs” carries out numerical experiments regarding boundary condition of direct iterative solution runs. This function explores the effect of boundary condition around a given reference case. This function records values of number of cells, boundary condition and initialization value variation, iteration metrics and solution metrics of each direct iterative solution run with varying boundary conditions. This function also saves the field contour of the final solution.

“explore_initialization_values_of_direct_runs” carries out numerical experiments regarding initialization value of direct iterative solution runs. This function explores the effect of initialization value around a given reference case. This function records values of number of cells, initialization value variation, iteration metrics and solution metrics of each direct iterative solution run with varying initialization value.

“explore_parameters_of_#_cycle”(#=V,W,F) carries out numerical experiments regarding parameters of multigrid cycles with fixed schemes. This function explores the effect of varying maximum coarse grid level and iteration per sweep around a reference case. This function records values of maximum coarse grid level, iteration per sweep, mean residual, reference work unit and cost ratios of each multigrid run with varying parameters. This function also saves the run summary plot of the multigrid run.

“explore_parameters_of_flexible_cycle” carries out numerical experiments regarding parameters of multigrid cycles with flexible schemes. This function explores the effect of varying alpha and beta parameters of flexible cycle around a reference case. This function records values of alpha, beta, mean residual, reference work unit and cost ratios of each multigrid run with varying parameters. This function also records a message if multigrid run indeed valid according to given operation limits. This function also saves the run summary plot of the multigrid run.

“explore_parameters_of_PID_driven_cycle” and “explore_criteria_of_PID_driven_cycle” carry out numerical experiments regarding parameters of multigrid cycles with PID Driven schemes. These functions explore the effect of varying PID coefficients and PID criteria of PID Driven cycle around a reference case, respectively. These functions record values of PID parameters (coefficients or criteria), mean residual, reference work unit and cost ratios of each multigrid run with varying parameters. This function also records a message if multigrid run indeed valid according to given operation limits. This function also saves the run summary plot of the multigrid run.

3.8.4. Result Generation Procedures

Sequential tasks are carried out in order to generate result packs that are presented in results section, to improve consistency and to eliminate human factor. These procedures are merely combinations of various functions and commands in a harmonized manner.

Procedure that generates validation result pack initially arranges titles, labels and other plot properties of various runs for given run data container. Analytical Laplace solution of the reference case is generated. With the addition of the exact solution, postprocess of each run are carried out. Field contour, residual scatter, residual progress, diagonal values, property distribution, error distribution, run summary and convergence level plots are generated. All data and plot are saved to respective directories.

Procedure that generates residual progress pack initially creates the residual progress graph of varying resolutions for further use in resolution exploration. Then, residual progress graph of direct iterative solution for given resolution is plotted with annotations. Similarly, first derivative of residual progress is also plotted. Field contour and residual scatter plots are generated for certain mean residual levels. Statistical analyses plots (property distribution, error distribution, box plot of property) of direct iterative runs are generated. Residual progress plots of multigrid runs are also generated. All data and plots are saved to respective directories.

Procedures that generate exploration result packs slightly differ from previous ones. Since exploration functions are also comprehensive procedures themselves, data required for exploration result packs are generated via exploration functions and then postprocess procedures are carried out. Results regarding the explorations of grid resolutions, boundary conditions and initialization values can be generated for direct iterative solution runs. Results regarding the explorations of different multigrid cycles can be generated for respective multigrid cycle types. Results of adaptive multigrid cycle schemes are treated differently. Parameter sets that end up with Invalid or infeasible multigrid runs prevented surface and contour representations of explorations; thus, scatter representations of data are generated. All data and plots are saved to respective directories.

4. RESULTS

Generating result packs to cover essential aspects of thesis was simply challenging. Therefore, results were presented with increasing complexity starting with validation of the tools (procedures) at disposal. Validation pack serves the answer to the question "do procedures get to correct solution?" by any means. Result pack of residual progress offers insights about "how procedures get to the solution?" anyway. Within the exploration pack, several aspects of procedures were discovered by changing input parameters and are presented in an informative manner. Parameter explorations were conducted to answer to questions "which parameters effect the way and/or the cost to solution?" as comprehensive as possible. Parameter explorations were planned according to principals of experimental design [43]. Statistical analyses of results are shared wherever applicable or appropriate.

4.1. Validation

Comparing outcome of the procedures to a reference outcome validated methods. Two-dimensional thermal diffusion problem on a homogenous plate was selected for the reference case which may have constant temperature and constant heat flux at boundaries. Selecting thermal diffusion case which has analytical Laplace solution as reference inherently enabled proper validation of methods. Run summary plots of the validation runs were also given in Appendix #5 with higher pixel resolution.

4.1.1. Reference Case

Thermal diffusion case has constant temperature and constant heat flux at boundaries [44]. Thermal coefficients and thickness are constant within the homogenous plate [44]. Reference case has certain dimensions on both directions [44]. Number of divisions of the domain were specifically selected to clearly present results before exploring higher grid resolutions.

Figure 4.1 was given to illustrate the reference case. Thermal conductivity was assumed constant as 1000 W/mK within the homogenous plate [44]. Thickness of the plate was assumed constant as 1cm [44]. Thermal diffusion case has constant temperature values at boundaries West, East, South and North as 0 °C, 0 °C, 0 °C and 100 °C respectively. Reference case has dimensions of 3m in both directions. Reference domain was generated with 80 division of cells in both directions.

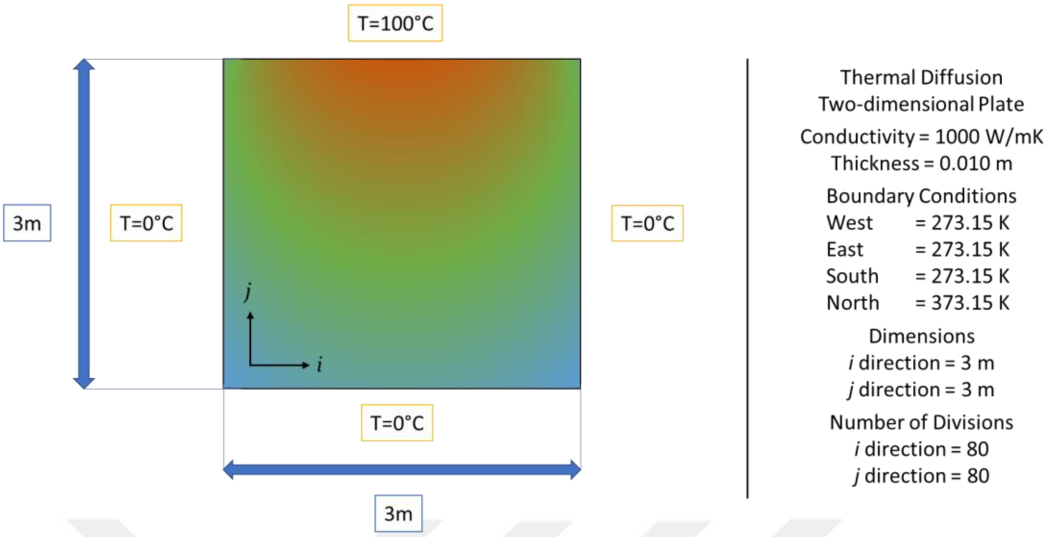


Figure 4.1. *Illustration of the reference case definition*

4.1.2. Analytical Laplace Solution

Thermal diffusion case can be solved by Laplace transform using boundary conditions [44]. Equation 4.3 was used to generate temperature solutions within the domain at any location [44].

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \quad for\ x = [0, a]\ and\ y = [0, b] \tag{4.1}$$

$$u(0, y) = 0\,°C\ ; u(a, y) = 0\,°C\ ; u(x, 0) = 0\,°C\ ; u(x, b) = 100\,°C \tag{4.2}$$

$$u(x, y) = \frac{400\,°C}{\pi} \sum_{k=1}^{\infty} \frac{\sin\left(\frac{n\pi x}{a}\right)\sinh\left(\frac{n\pi y}{b}\right)}{n sinh(n\pi)} \quad for\ n = 2k - 1 \tag{4.3}$$

Field contour plot was given in Figure 4.2 to illustrate exact solution of the case. Since solution is exact and residual is practically zero; analytical solution of the case can be used to measure any error within the domain which are generated by other numerical methods.

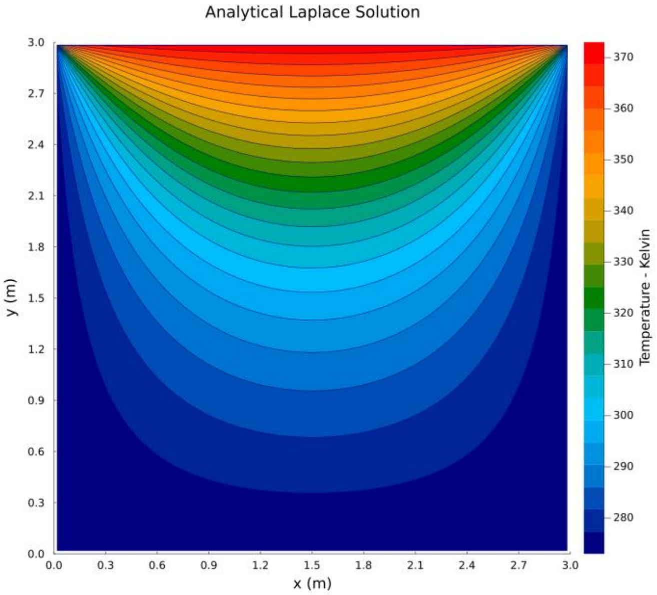


Figure 4.2. *Analytical Laplace solution field contour plot of the reference case*

4.1.3. Direct Iterative Solution

Reference case was solved via Gauss-Seidel iterative method as described after preprocessing the input by finite volume method functions. Constant initial guess value was given to the domain. Constant initial guess value was calculated as 298.15 °K by averaging temperature values of boundary conditions. Convergence criteria was given as 0.001 (°K) mean residual and was reached after 1028 iterations.

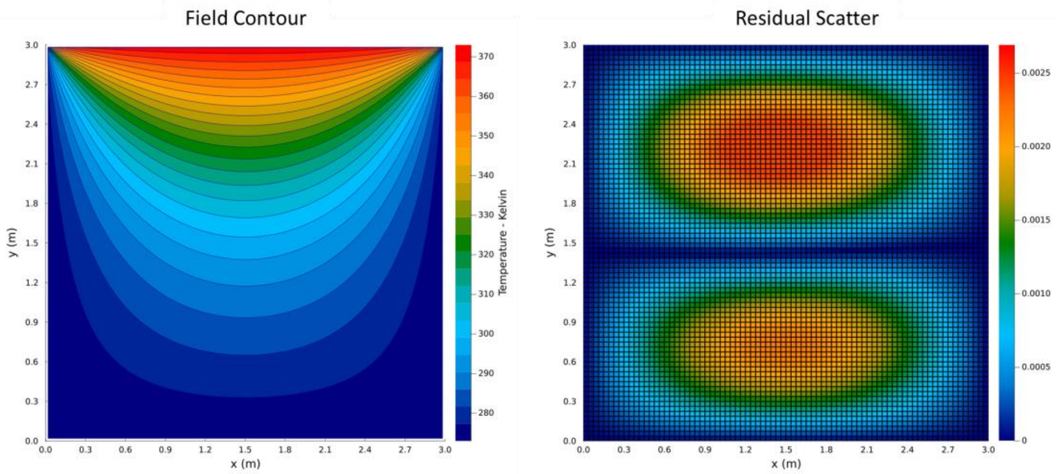


Figure 4.3. *Direct iterative solution field contour plot (left) and residual scatter (right) plot of the reference case.*

4.1.4. Fixed Cycle Solutions

Reference case was solved via multigrid cycles with fixed schemes, namely V cycle, W cycle and F cycle. Constant initial guess value was calculated as 298.15 °K by averaging temperature values of boundary conditions and was given to the domain. Iteration number per sweep on each resolution was constant and was given as 5 for V cycle, W cycle and F cycle. Maximum coarsening level was 4 for V, W and F cycles.

Field contour plots and residual scatter plots were given in Figure 4.4, Figure 4.5 and Figure 4.6 to illustrate multigrid cycle solutions and residual distributions of the reference case. Summary of multigrid cycle runs were given for V, W and F cycles in Figure 4.7, Figure 4.8 and Figure 4.9 respectively to illustrate operations and steps that solution gets through. Convergence levels of solutions were 0.00117, 0.00122, 0.00122 (°K) mean residual for V, W and F cycles respectively. Even though most of the domain was converged fairly, as seen in Figure 4.4, Figure 4.5 and Figure 4.6; multigrid cycles with fixed schemes generated high residual values for cells with high gradients of temperature.

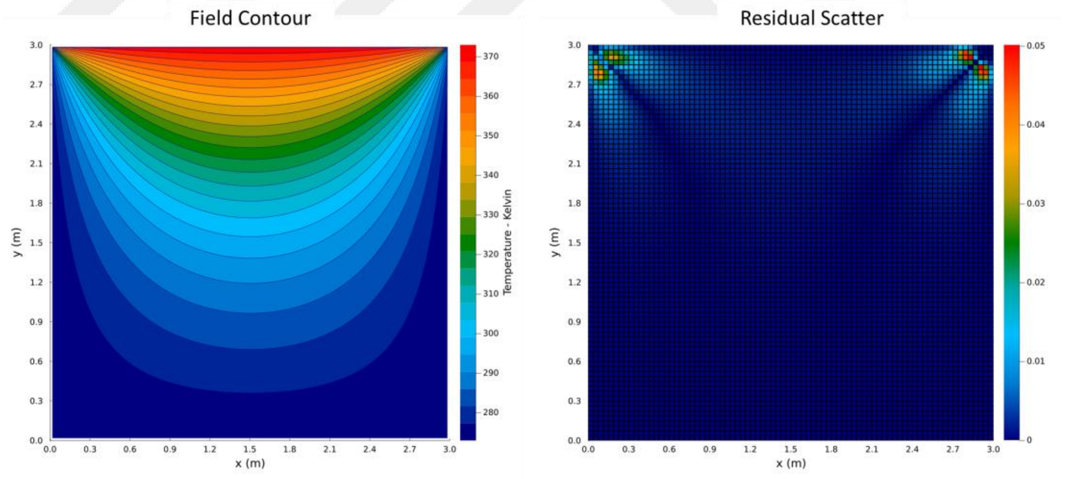


Figure 4.4. *Multigrid V cycle solution field contour plot (left) and residual scatter (right) plot of the reference case*

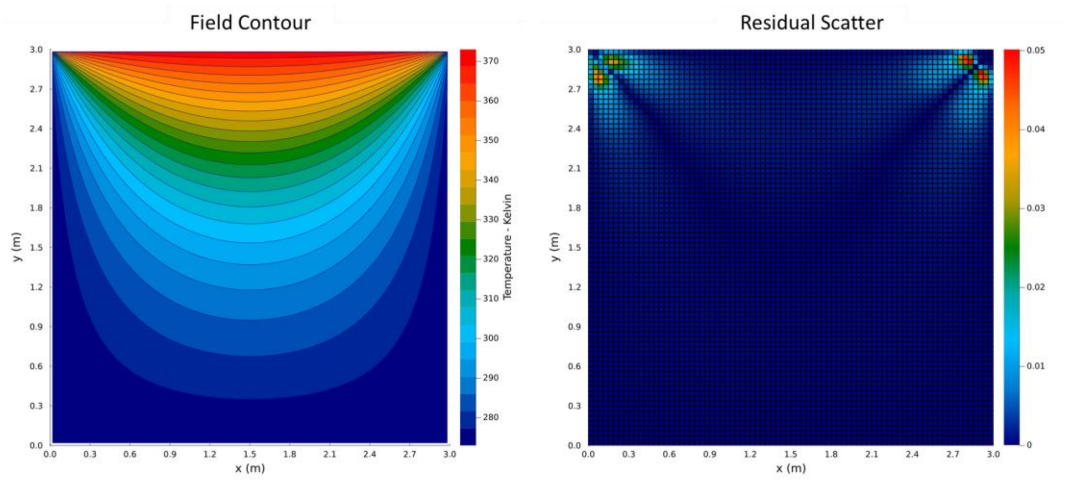


Figure 4.5. *Multigrid W cycle solution field contour plot (left) and residual scatter (right) plot of the reference case*

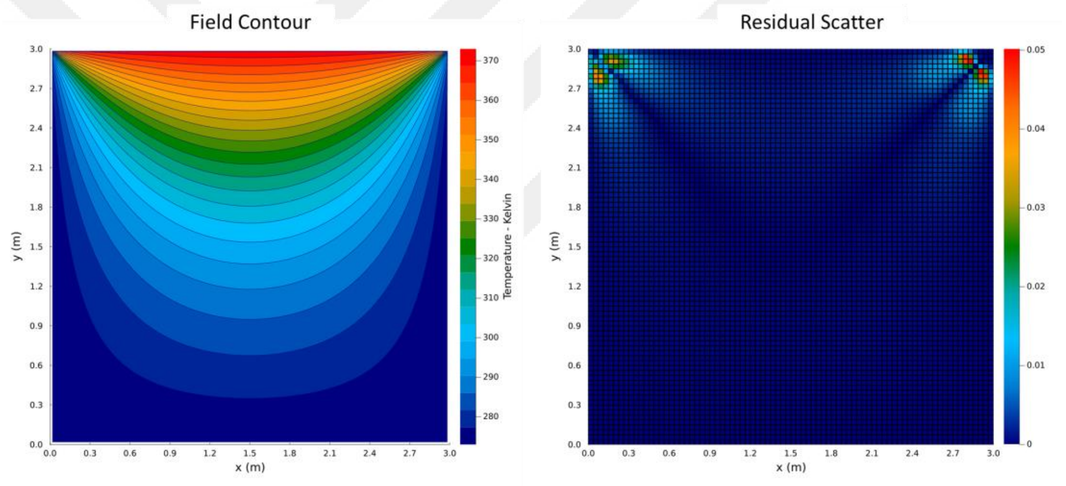


Figure 4.6. *Multigrid F cycle solution field contour plot (left) and residual scatter (right) plot of the reference case*

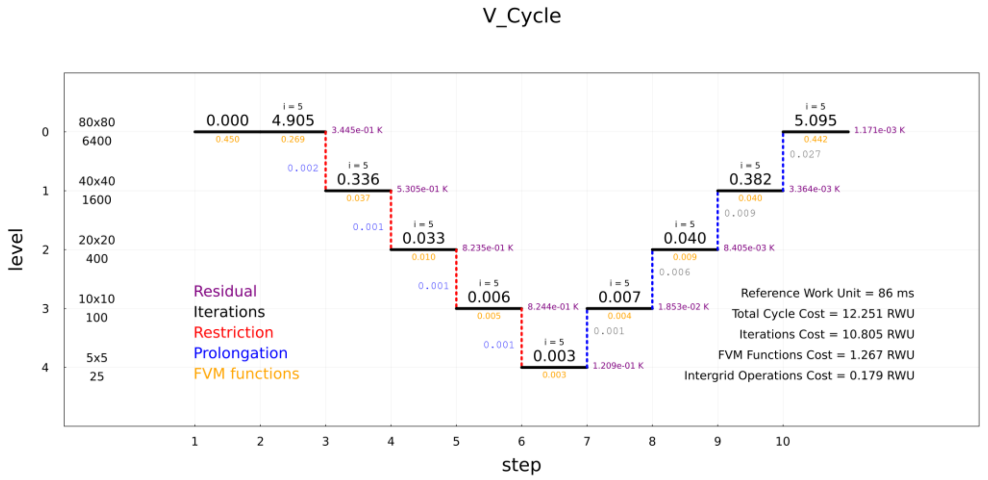


Figure 4.7. *Run summary of multigrid V cycle solution of the reference case*

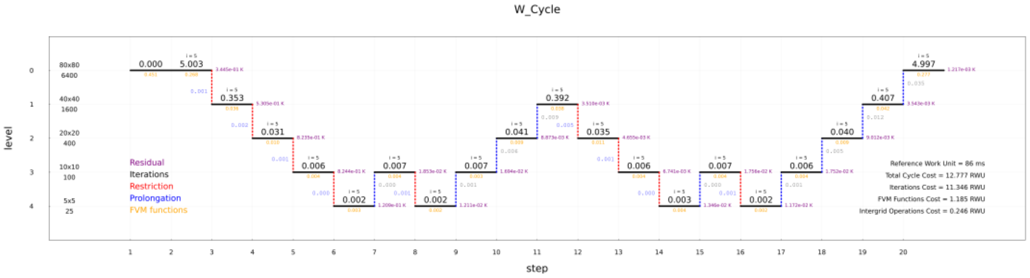


Figure 4.8. *Run summary of multigrid W cycle solution of the reference case*

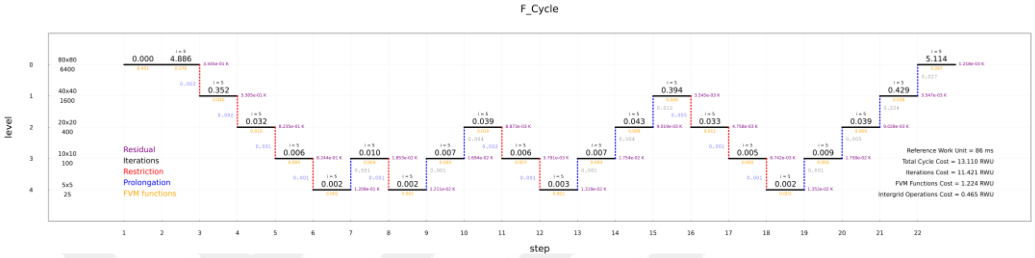


Figure 4.9. *Run summary of multigrid F cycle solution of the reference case*

4.1.5. Flexible Cycle Solution

Reference case was solved via flexible multigrid cycle with adaptive scheme. Constant initial guess value was calculated as 298.15 °K by averaging temperature values of boundary conditions and was given to the domain. There was no constant iteration number per sweep on each resolution. Flexible cycle runs with alpha and beta parameters which controls the steering between resolutions. Thus, iteration numbers per sweep on each resolution were fallout of circumstances and had a limit of 2 at minimum. Cycle inputs were given; alpha as 0.2 and beta as 0.8 for the reference condition. Convergence criteria was given as 0.001 (°K) mean residual.

Field contour plots and residual scatter plots were given in Figure 4.10 to illustrate flexible multigrid cycle solution and residual distributions of the reference case. Summary of flexible multigrid cycle run was given in Figure 4.11 to illustrate operations and steps that solution gets through. Convergence level of solution was 0.00078 (°K) mean residual and was reached after 14 steps with 1 preprocess step on initial resolution and 13 sweeps on various resolutions. Even though most of the domain was converged fairly, as seen in Figure 4.10, flexible multigrid cycle also generates high residual values for cells with high gradients of temperature as multigrid cycles with fixed schemes. Additionally, convergence level of the solution was more precise than the convergence target input which can be inferred as that flexible multigrid cycle not only ensured but also exceeded convergence target.

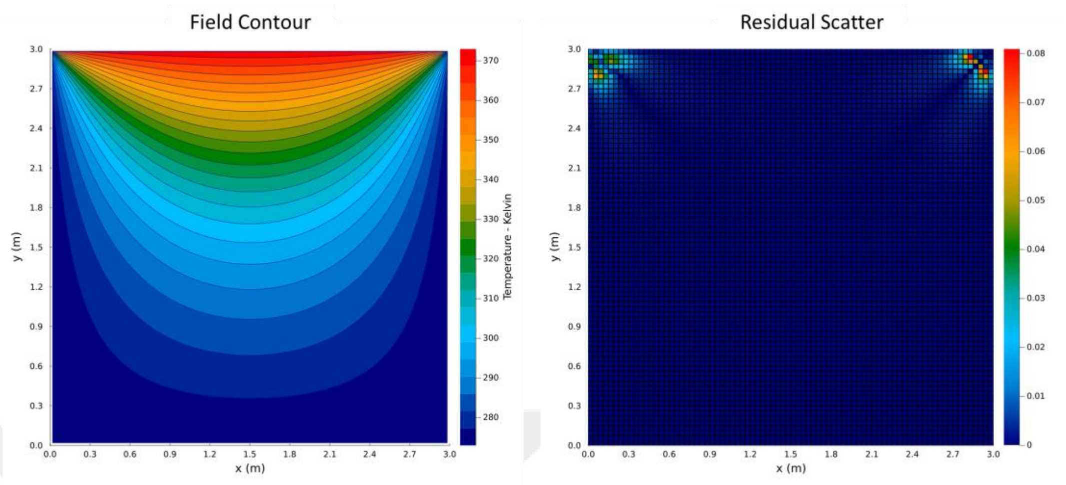


Figure 4.10. *Flexible multigrid cycle solution field contour plot (left) and residual scatter (right) plot of the reference case*

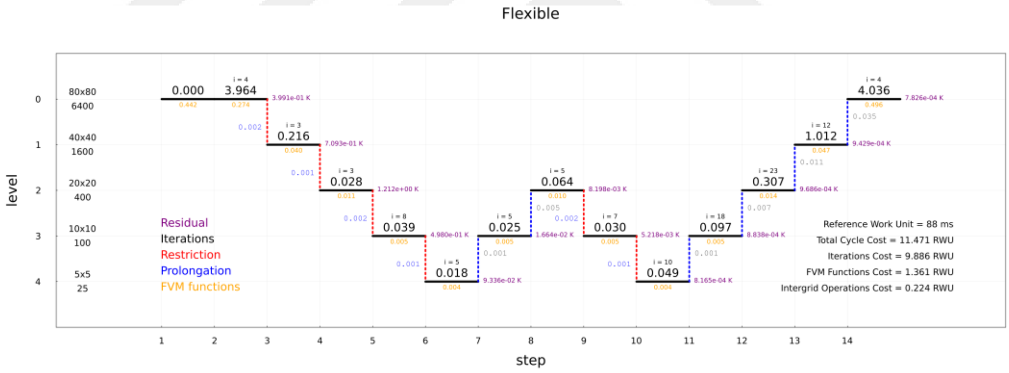


Figure 4.11. *Run summary of flexible multigrid cycle solution of the reference case*

4.1.6. PID Driven Cycle Solution

Reference case was solved via PID driven multigrid cycle with adaptive scheme. Constant initial guess value was calculated as 298.15 °K by averaging temperature values of boundary conditions and was given to the domain. There was no constant iteration number per sweep on each resolution. PID driven cycle runs with cP, cI, cD parameters Pc, Ic, Dc criteria which controls the steering between resolutions. Thus, iteration numbers per sweep on each resolution were fallout of circumstances and had a limit of 2 at minimum. Cycle inputs were given; cP, as 1.0, cI as 1.0, cD as 1.0, Pc as 0.2, Ic as 5.0, Dc as -0.1 for the reference conditions. Convergence criteria was given as 0.001 (°K) mean residual.

Field contour plots and residual scatter plots were given in Figure 4.12 to illustrate PID driven multigrid cycle solution and residual distributions of the reference case. Summary of PID driven multigrid cycle run was given in Figure 4.13 to illustrate operations and steps that solution gets through. Convergence level of solution was 0.00090 (°K) mean residual and was reached after 22 steps with 1 preprocess step on initial resolution and 21 sweeps on various resolutions. Even though most of the domain was converged fairly, as seen in Figure 4.12, PID driven multigrid cycle too generated high residual values for cells with high gradients of temperature as multigrid cycles with fixed schemes. Additionally, convergence level of the solution was more precise than the convergence target input which can be inferred as that PID driven multigrid cycle also not only ensured but exceeded convergence target.

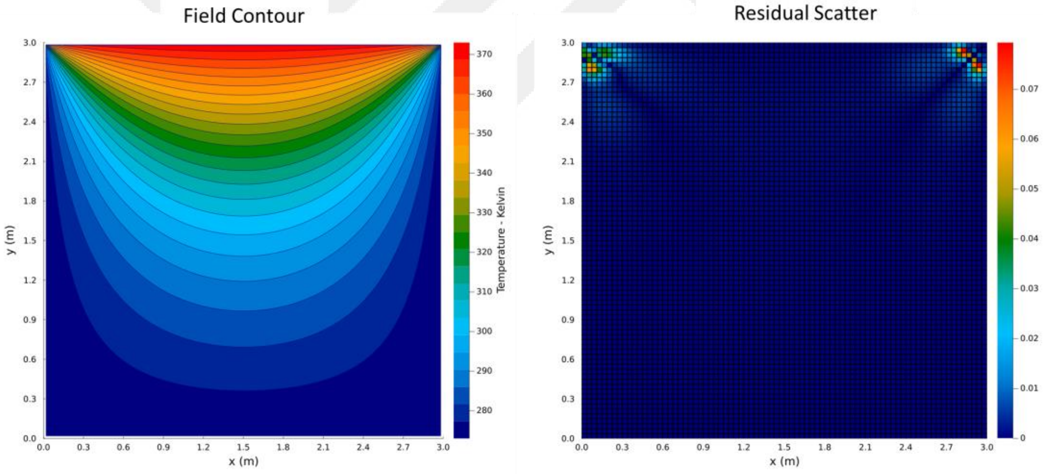


Figure 4.12. *PID driven multigrid cycle solution field contour plot (left) and residual scatter (right) plot of the reference case*

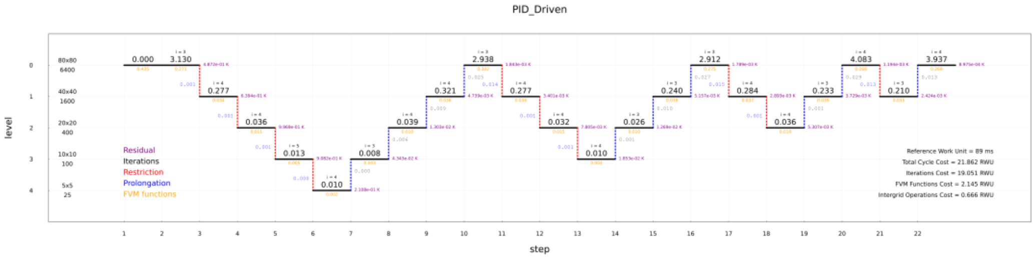


Figure 4.13. *Run summary of PID driven multigrid cycle solution of the reference case*

4.1.7. Comparison of Solutions

Visual inspections of field contour plots of solutions gave the impression that all solutions were satisfactory. However, residual scatter plots had apparent differences between direct iterative solutions and multigrid cycle solutions. In order to properly find out if solutions were in fact converged and were correct, temperature values at a diagonal line within domain were read. Figure 4.14 was generated to compare numerical solutions against analytical solution.

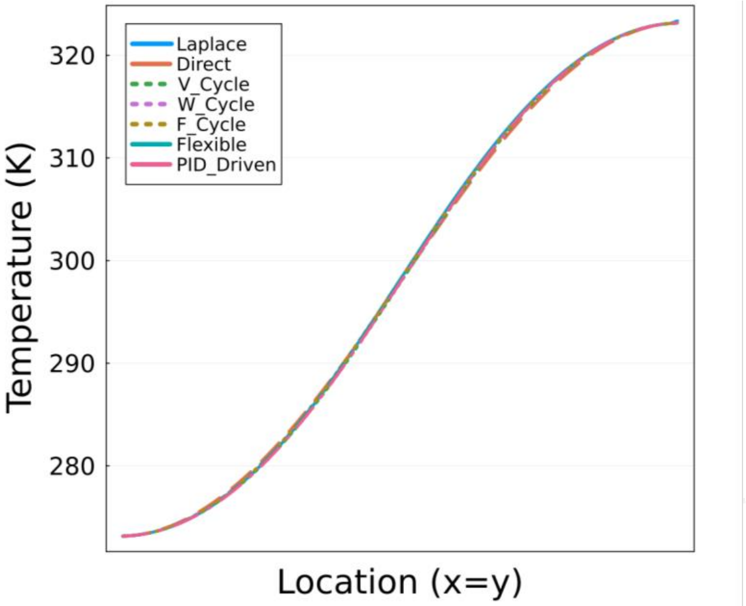


Figure 4.14. *Comparison of temperature values on a diagonal line within domain for various runs.*

Satisfactory solution accuracies were observed for all solutions while multigrid cycle solutions having exceptional proximity to analytical solution. Although Figure 4.14 alone validated methods towards an accurate solution generation, further statistical analyses were carried out in order to get comprehensive insight of the situation. Distribution (density) curves of temperature values within the domain were generated and were given in Figure 4.15 for inspection. Exact error distribution (density) curves within the domain were generated and were given in Figure 4.16 for inspection. Box plots that notify mean, extrema and quartile values were also generated and were given in Figure 4.17 for temperature values within the domain. Convergence levels of each solution were visualized in Figure 4.18 to remind mean residual values of solutions.

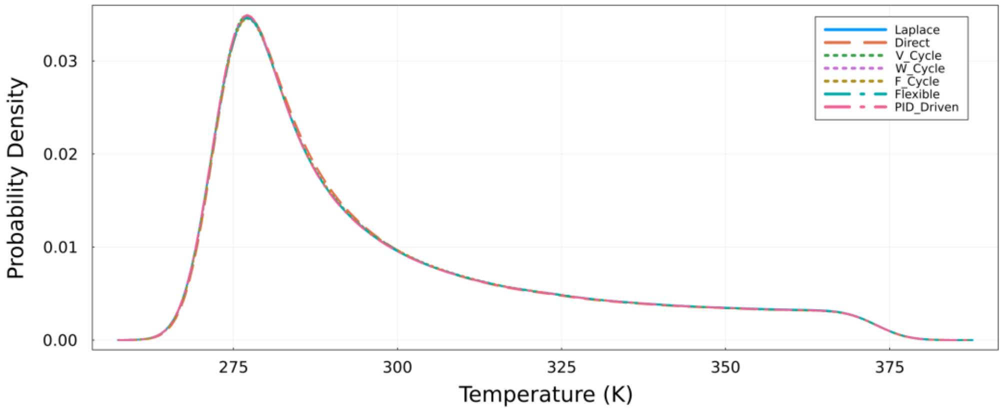


Figure 4.15. *Comparison of density curves of temperature values within domain for various runs*

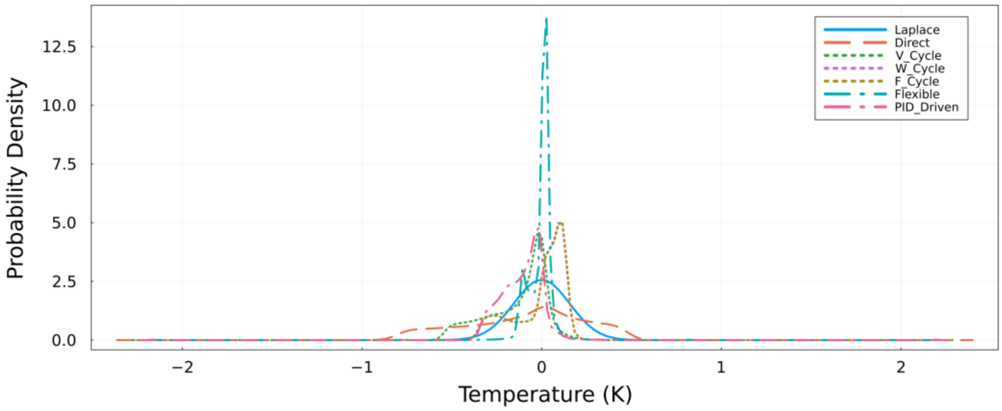


Figure 4.16. *Comparison of density curves of exact error within domain for various runs*

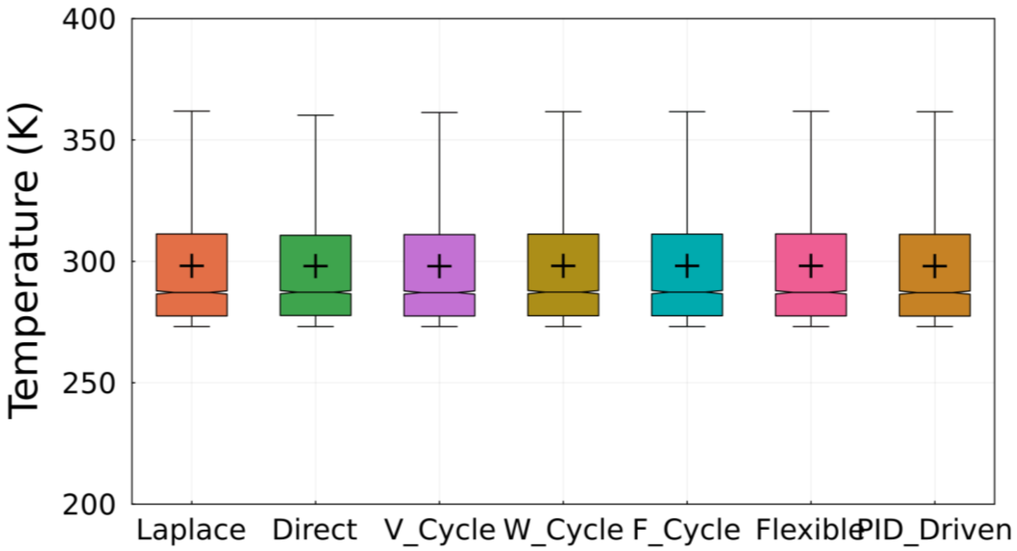


Figure 4.17. *Comparison of box plots of temperature values within domain for various runs*

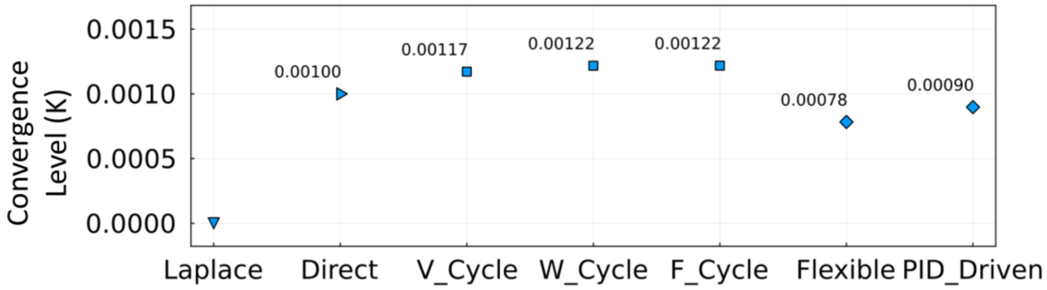


Figure 4.18. *Comparison of convergence levels in terms of mean residual values for various runs*

Analyses of data and plotting figures revealed that for a certain mean residual level of solution, other statistical parameters were aligned as well. Fair amount of investment was required to get fairly consistent outcome with high precision convergence of mean residual levels amongst various solutions. Thus, further examination of residual progress and/or behaviour was required to understand solution process.

4.2. Residual Progress

Mean residual is an essential concept that is conventionally monitored during computational analyses [1]. For high precision convergence values, it was observed that monitoring mean residual in fact would work towards realizing a correct solution. However, the path undertaken to get such correct solution should be examined in order to truly reveal residual behaviour. Cost of the path should also be a subject of consideration. Examining residual progress can also shed light to opportunities towards cost reduction as well as complimentary concepts that would ensure getting a correct solution.

4.2.1. Direct Iterations

Distributions of residuals within the domain were examined for several convergence levels of mean residual for the direct iterative solution run. Residual progress curve was generated and was presented in Figure 4.19 in which iteration counts were marked for certain convergence levels. Figure 4.21 was composed to visualize solution and residual distribution progress with mean residual convergence.

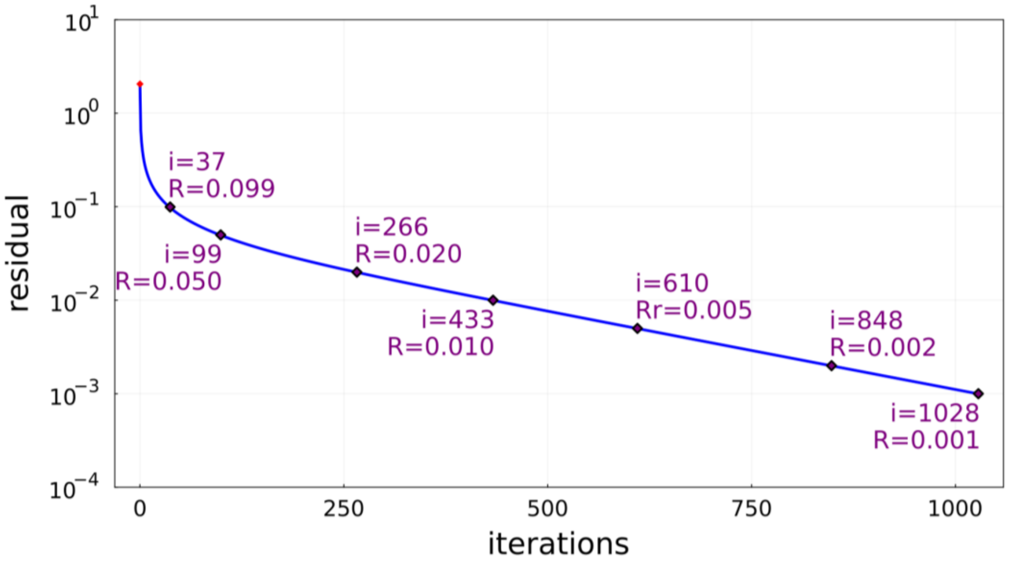


Figure 4.19. *Residual progress curve of direct iterative solution*

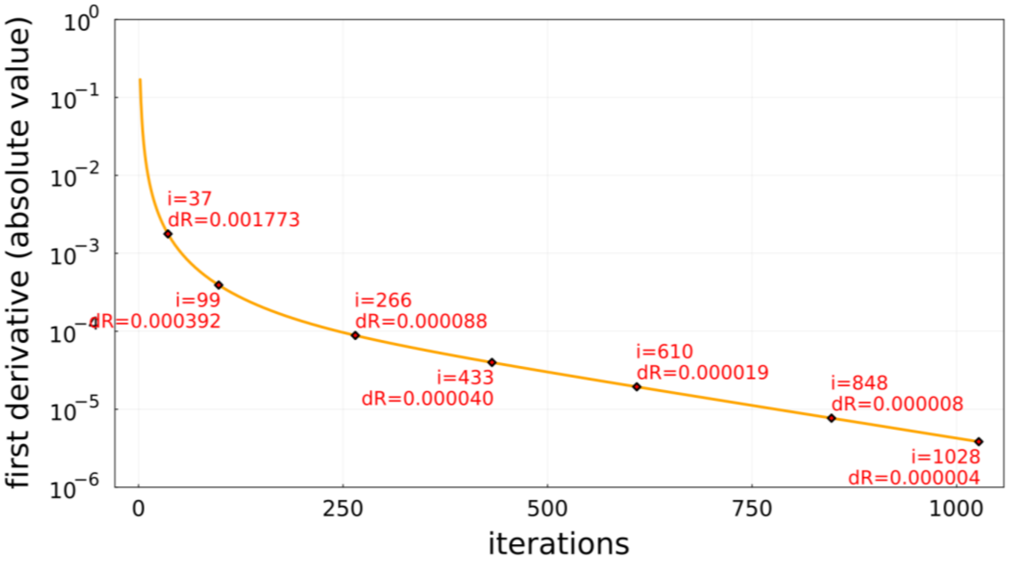


Figure 4.20. *First derivative of residual progress curve of direct iterative solution*

First derivative of residual curve with absolute values were given in Figure 4.20 to emphasize the progress towards solution. Inspection of Figure 4.20 clearly showed that improvement (change) with each iteration diminishes to below 0.0001 per iteration after approximately 250 iterations. Even though rate of change diminished, solution was not satisfactory as seen in Figure 4.21. In order to achieve correct solution, iterations continued till convergence target of mean residual was reached at a grinding rate of improvement for each iteration.

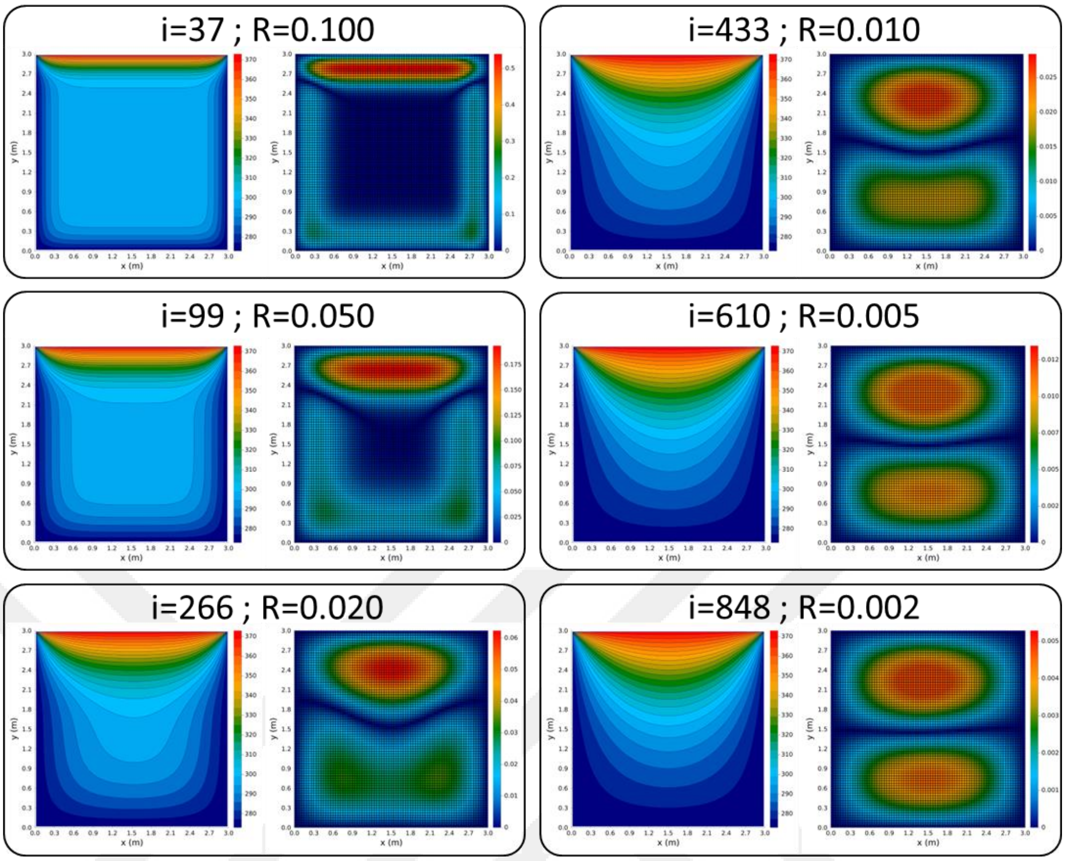


Figure 4.21. *Solution progress by field contour and residual scatter plots*

Inspecting Figure 4.21 leaded to other observations on solution progress through the iterative process. Solution accuracy was clearly unacceptable before mean residual was at 0.01 levels. Even then, residual scatter within domain did not seem converged. Thus, further statistical analysis was required in order to understand when to end iterative process to get converged and correct solution.

Temperature values on a diagonal line as seen in Figure 4.22 inferred that solution accuracy was not acceptable before the convergence levels of 0.005 for mean residual. Temperature distribution curves in Figure 4.23 and exact error distribution curves in Figure 4.24 also supported the indication inferred from Figure 4.22 which was that solution accuracy was not satisfactory before certain convergence level of mean residual. Boxplots given in Figure 4.25 revealed that quartiles of temperature values within domain were indeed not converged till certain convergence of mean residual. In other words, to get an accurate solution, convergence of mean residual was ambiguous at best, while converged quartiles alongside mean residual assured satisfactory results.

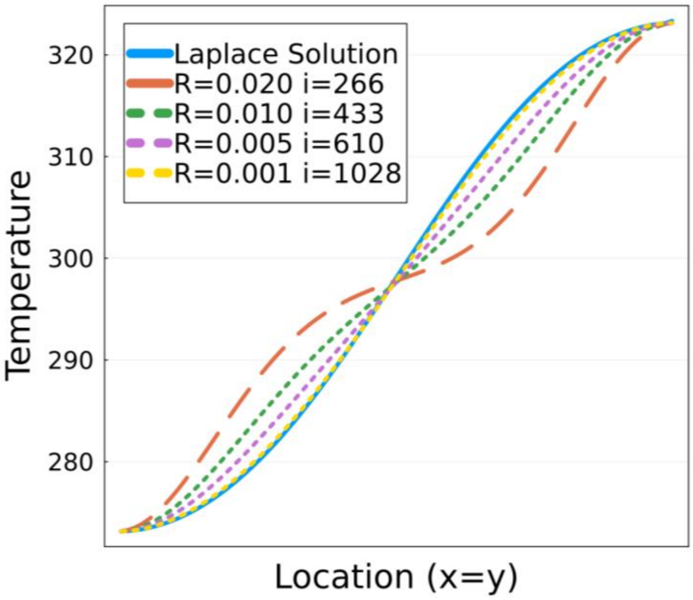


Figure 4.22. *Comparison of temperature values on a diagonal line within domain for various convergence levels of mean residual*

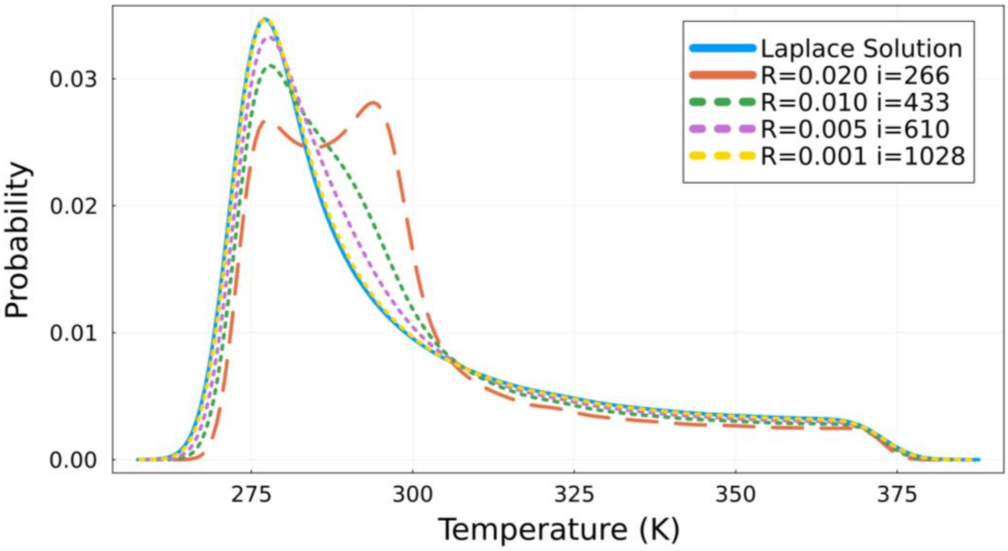


Figure 4.23. *Comparison of density curves of temperature values within domain for various convergence levels of mean residual*

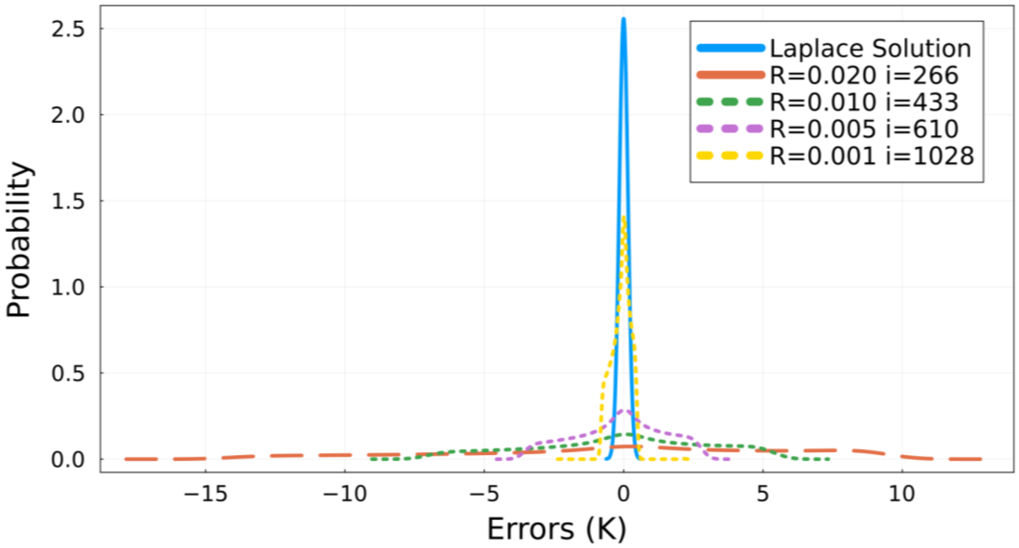


Figure 4.24. *Comparison of density curves of exact error within domain for various convergence levels of mean residual*

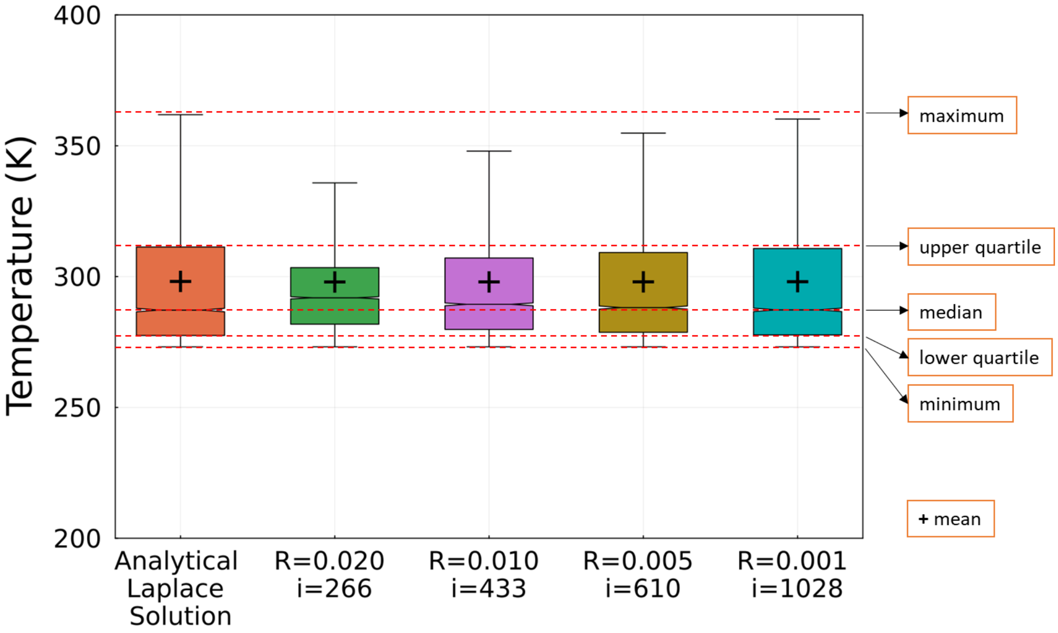


Figure 4.25. *Comparison of box plots of temperature values within domain for various convergence levels of mean residual*

4.2.2. Fixed Cycles

Residual progress curves of multigrid cycles with fixed schemes were generated and presented in Figure 4.26, Figure 4.27 and 4.28 for V, W and F multigrid cycles respectively. Residual progress curves of multigrid cycles differed from direct iterative counterparts by having gaps between steps because of intergrid (grid resolution change) operations. Thus, each step and cumulative progress were plotted against operation and/or iteration count. Field contour plots and residual scatter plots were already given in Figure 4.4, Figure 4.5 and 4.6 to illustrate solution for V, W and F multigrid cycles respectively.

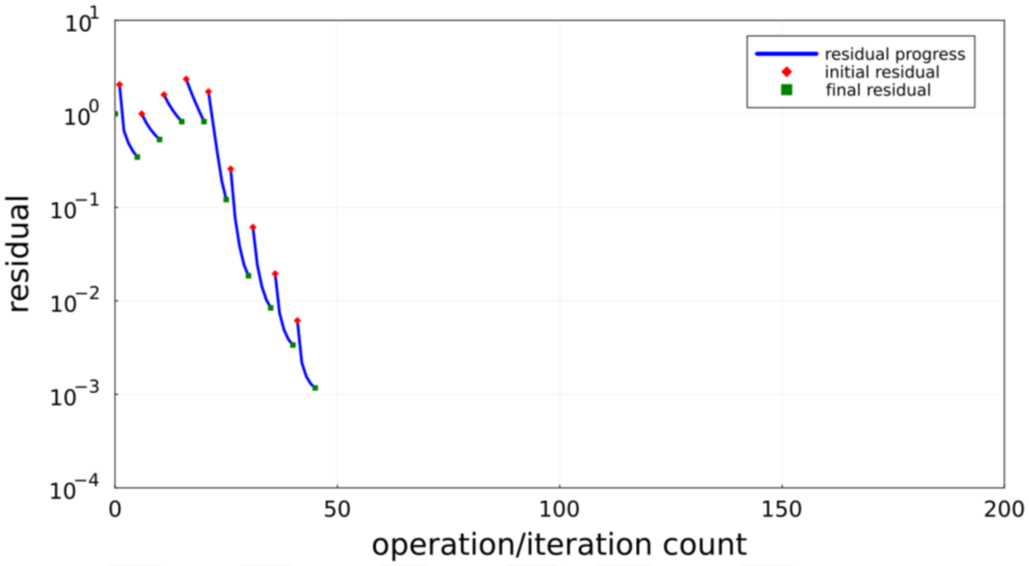


Figure 4.26. *Residual progress curve of multigrid V cycle solution*

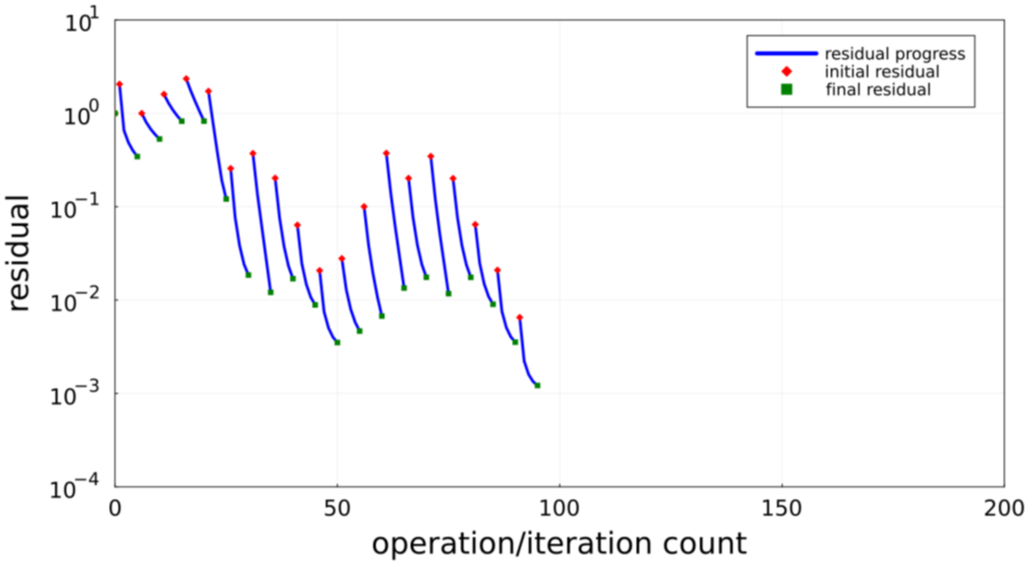


Figure 4.27. *Residual progress curve of multigrid W cycle solution*

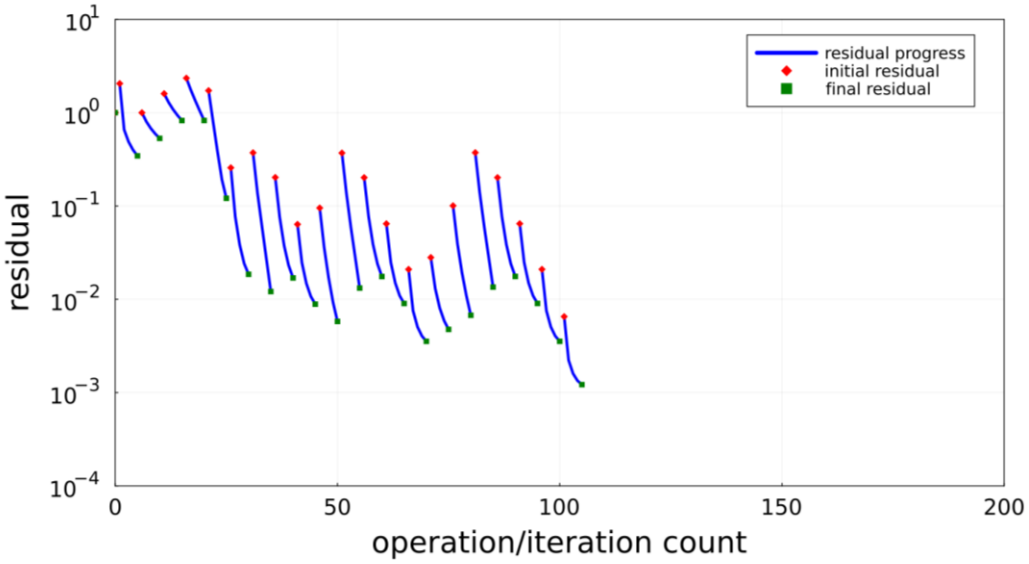


Figure 4.28. *Residual progress curve of multigrid F cycle solution*

4.2.3. Flexible Cycle

Residual progress curve of flexible multigrid cycle with adaptive scheme was generated and presented in Figure 4.29 to be inspected. Field contour plots and residual scatter plots were already given in Figure 4.10 to illustrate solution for flexible multigrid cycle.

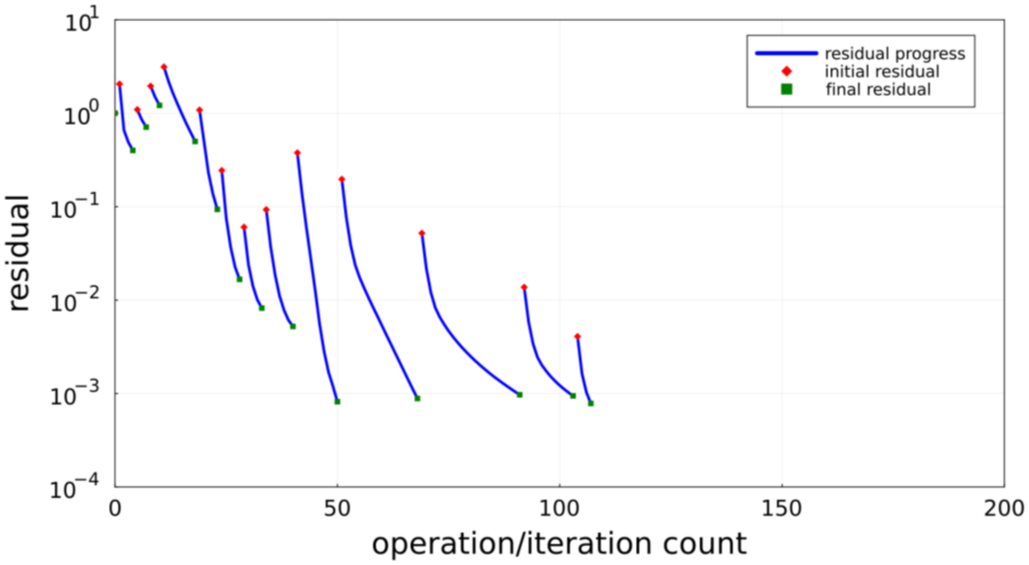


Figure 4.29. *Residual progress curve of flexible multigrid cycle solution*

4.2.4. PID Driven Cycle

Residual progress curve of PID driven multigrid cycle with adaptive scheme was generated and presented in Figure 4.30 to be inspected. Field contour plots and residual scatter plots were already given in Figure 4.10 to illustrate solution for PID driven multigrid cycle.

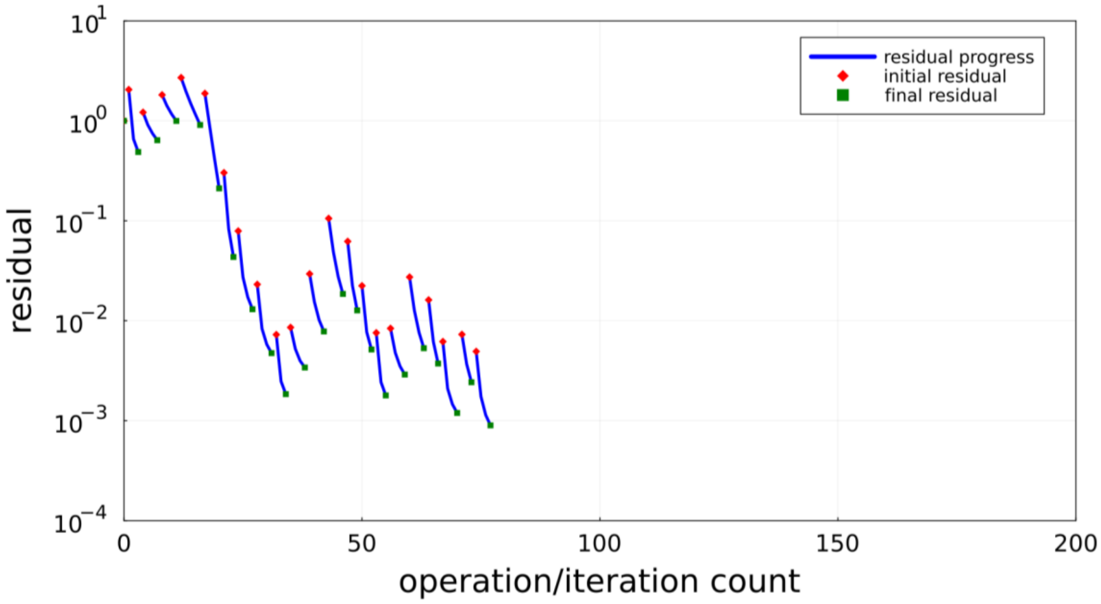


Figure 4.30. *Residual progress curve of PID driven multigrid cycle solution*

4.2.5. Comparison of Cycles

Validation of methods and residual progress of direct iterative runs showed that solution was acceptable at convergence levels of 0.001 for mean residual. Final convergence levels of each multigrid cycle as seen in Figure 4.18 were approximately at same level. In this manner, residual scatter plots were generated and presented in Figure 4.31 in order to fairly compare residual distribution for each solution. Colour scales of plots were kept limited between 0.0 to 0.001 residual values in Figure 4.31. Cells with colours which were not red in Figure 4.31 are converged below 0.001 level of residual value.

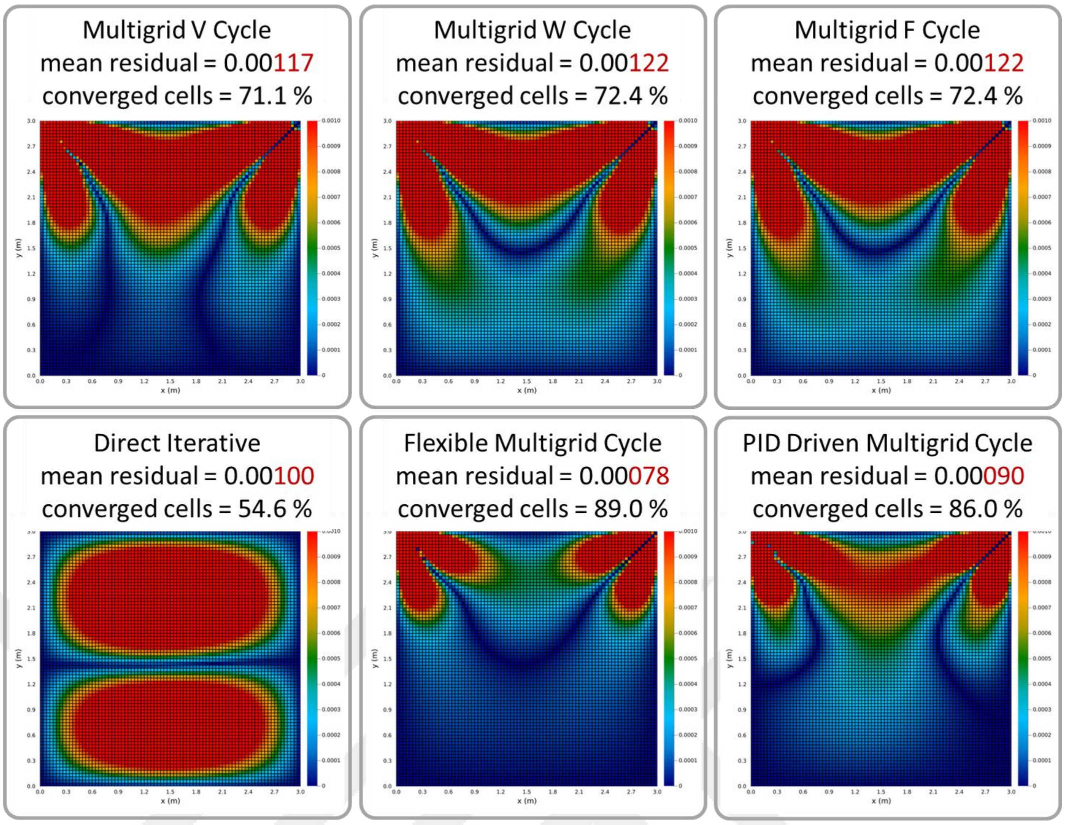


Figure 4.31. *Residual scatter plots comparison of solutions for similar convergence of mean residual*

Crucial observations can be inferred from Figure 4.31 regarding residual progress of a solution. First of all, multigrid cycles performed significantly better compared to direct iterative method in terms of residual distribution while having approximately similar mean residual levels. Secondly, multigrid cycle solutions had high residual values at locations with high gradient of temperature change while direct iterative method did not. Multigrid V, W and F cycle solutions appeared comparably same regarding residual distributions as well as mean residual levels. Flexible and PID driven multigrid cycles appeared relatively similar regarding residual distributions as well as mean residual levels. Main difference between V cycle and other fixed schemes was the number of steps undertaken on coarse grid levels. W and F cycles had much more steps on coarse grid levels than V cycle as seen in run summary illustrations given in Figure 4.7, Figure 4.8 and Figure 4.9. Main difference between flexible and PID driven cycles were conceptually same. PID driven cycle visited fine grid resolutions more than flexible cycle which went through more iterations on coarse grid levels as seen in run summary illustrations given in Figure 4.11 and Figure 4.13.

Numbers regarding cost of each solution at 0.001 mean residual level were summarized in Table 4.1. Reference work unit is the elapsed time for a single iteration on finest grid and was measured for each solution on the run. Table 4.1 should be read by reminding that; V cycle, W cycle and F cycles had 5 iterations per sweep, flexible cycle had alpha as 0.2 and beta as 0.8 for input parameters, PID driven cycle had cP, as 1.0, cI as 1.0, cD as 1.0, Pc as 0.2, Ic as 5.0, Dc as -0.1 for input parameters. All of the input parameters were subject to change in parameter explorations. Since performance and/or cost of multigrid cycles are dependent on input parameters, solution of the reference case with initial input parameters should be recognized as establishing a reference foundation for further studies.

Table 4.1. *Cost information of each solution at 0.001 mean residual level.*

	Direct Iterative	V Cycle	W Cycle	F Cycle	Flexible Cycle	PID driven Cycle
(Operation, Iteration) Count	(1,1028)	(10,45)	(20,95)	(22,105)	(14,107)	(22,77)
Mean Residual Convergence Level	0.001	0.00117	0.00122	0.00122	0.00078	0.0009
Converged Cell Number Percentage	54.6%	71.1%	72.4%	72.4%	89.0%	86.0%
Cost in Reference Work Units	1028	11.9	12.9	12.6	11.0	21.5
Cost with respect to Direct Iterative	100.00%	1.16%	1.25%	1.23%	1.07%	2.09%

Almost all multigrid cycles delivered same if not better solution accuracy compared to direct iterative method, for only approximately 1% cost of the direct iterative solution. It is imperative to remind that cost ratio of multigrid cycles would vary with input parameters as well as case parameters and circumstances.

According to insights revealed by Figure 4.31 and Table 4.1, high performing multigrid cycle should minimize iteration count on fine grid resolutions and maximize iteration count on coarse grid resolutions. Furthermore, convergence should be monitored not only with mean residual levels, but with additional concepts such as quantiles of distributions of physical properties. Thus, it was necessary to conduct parameter explorations, since validation and progress of solutions for reference case were adequately revealed.

4.3. Explorations

Validation of methods and solution progress of various runs, indicated the necessity of exploration of effects and/or outcome of input parameters and/or variables. Parameter exploration should reveal how the way and/or the cost to solution are affected from variables.

All methods (direct iterative and multigrid cycles) were able to produce valid solutions with appropriate input parameters. Fair amount of cost was required to achieve a valid solution for direct iterative solutions. Multigrid cycles reduced costs to incredible levels. There were essential differences between direct iterative solutions and multigrid cycle solutions regarding residual distribution. Quartile progress of flow properties (temperature for the reference case) should be investigated.

Investigation of grid resolution and initialization value should help determine the reference input values towards multigrid cycle explorations. Thus, grid resolutions that are product of consecutive intergrid operations were selected. Restriction operations would take domain from the finest grid to coarsest grid within selected resolutions. Vice versa, prolongation operations would take domain from the coarsest grid to finest grid within selected resolutions.

Multigrid cycles with fixed schemes should be explored around iteration per sweep and maximum coarsening level while monitoring cost and convergence level as outcome. Multigrid cycles with adaptive schemes (flexible and PID driven) should be investigated around their input parameters while monitoring cost. Adaptive schemes require at least two iterations per sweep in order to calculate rate of change regarding either to monitor steering criteria or convergence criteria. Flexible cycle should be explored around alpha and beta parameters. PID driven cycle should be explored more thoroughly, around constants of proportional, integral and derivative terms (cP, cI, cD) as well as constants of proportional, integral and derivative steering criteria (Pc, Ic, Dc) to get a proper impression. PID driven multigrid cycle may have much more interaction between input parameters compared to flexible multigrid cycle, which imposes more risks and opportunities on the outcome.

4.3.1. Grid Resolutions

Mean residual progress and property (temperature) quartile progress on different grid resolutions were inspected. Metrics of direct iterative solutions were given in Table

4.2. All solutions had practically same convergence level of mean residual. Iteration count and time cost to achieve a valid solution increased with increasing grid resolution. Figure 4.32 were given to illustrate mean residual progress with respect to iterations.

Table 4.2. *Direct iterative solution metrics for different grid resolutions*

Number of Divisions	Number of Cells	Iteration Count	Solution Time	Convergence Level
5 x 5	25	21	0.0030 s	9.541E-04 (°K)
10 x 10	100	61	0.013 s	9.611E-04 (°K)
20 x 20	400	158	0.15 s	9.830E-04 (°K)
40 x 40	1600	358	2.6 s	9.929E-04 (°K)
80 x 80	6400	1028	92.4 s	9.999E-04 (°K)
160 x 160	25600	2653	3467 s	9.994E-04 (°K)

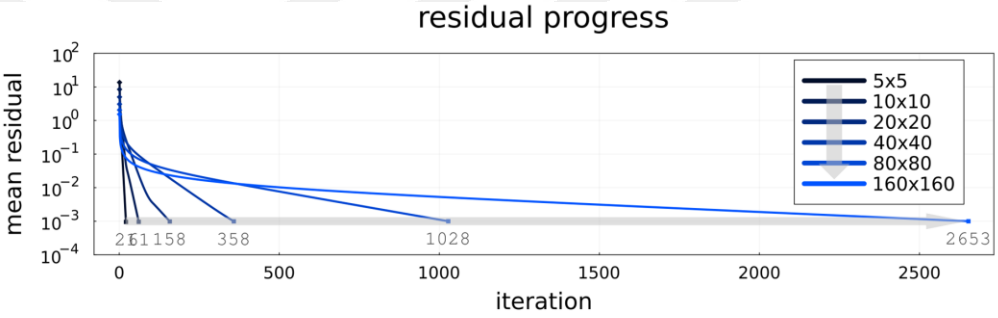


Figure 4.32. *Residual progress curves of solutions with different grid resolutions*

Mean residual progress started with high rate of change, continued with diminishing rate of change and end with creeping rate of change on fine grids. Mean residual progress started with high rate of change and kept rate of change throughout the iterative process on coarse grids.

Temperature distribution progress curves were given in Figure 4.33 with mean and quartile values supported with analytical solution counterparts. Although differences in mean value progresses were not apparent at first sight, quartile value progresses were clearly observed for different grid resolutions. Quartile values converged rapidly on coarse grids and did not even converge on finest grids. In terms of capturing property distribution across the solution domain, coarse grids exceptionally outperformed fine grids relatively.

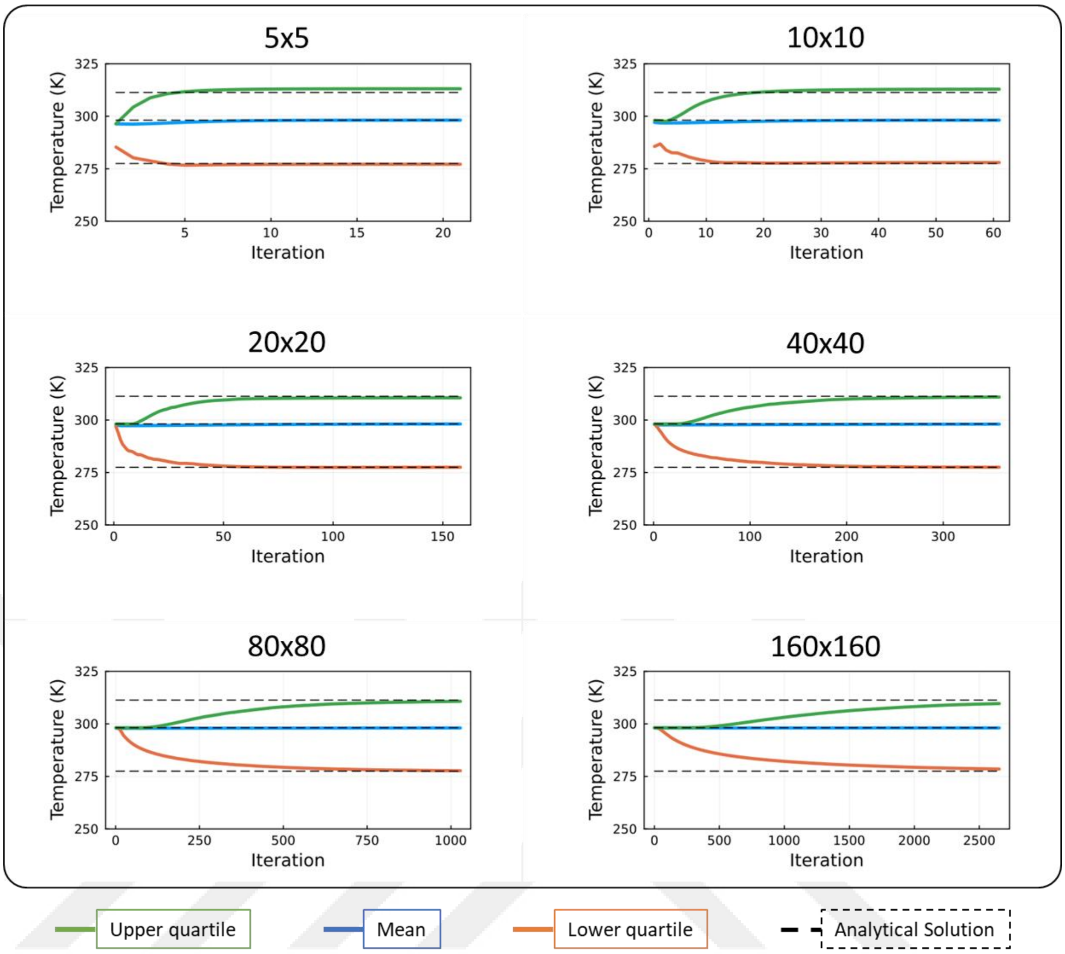


Figure 4.33. *Temperature mean and quartile progress curves of solutions with different grid resolutions*

Cost of each iteration throughout the solution process were visualized in Figure 4.34 for each grid resolution. Elapsed time for each iteration was measured with time stamp differences between at the start and end of the iteration. Time cost of iterations unpredictably varied for each grid resolution with a slightly increasing trend as the iterations continue. Even with the standard deviation filter, variations and spikes on curves were present. Although time cost per iteration varied around a mean value, individual estimation of an iteration cost was seemed not possible.

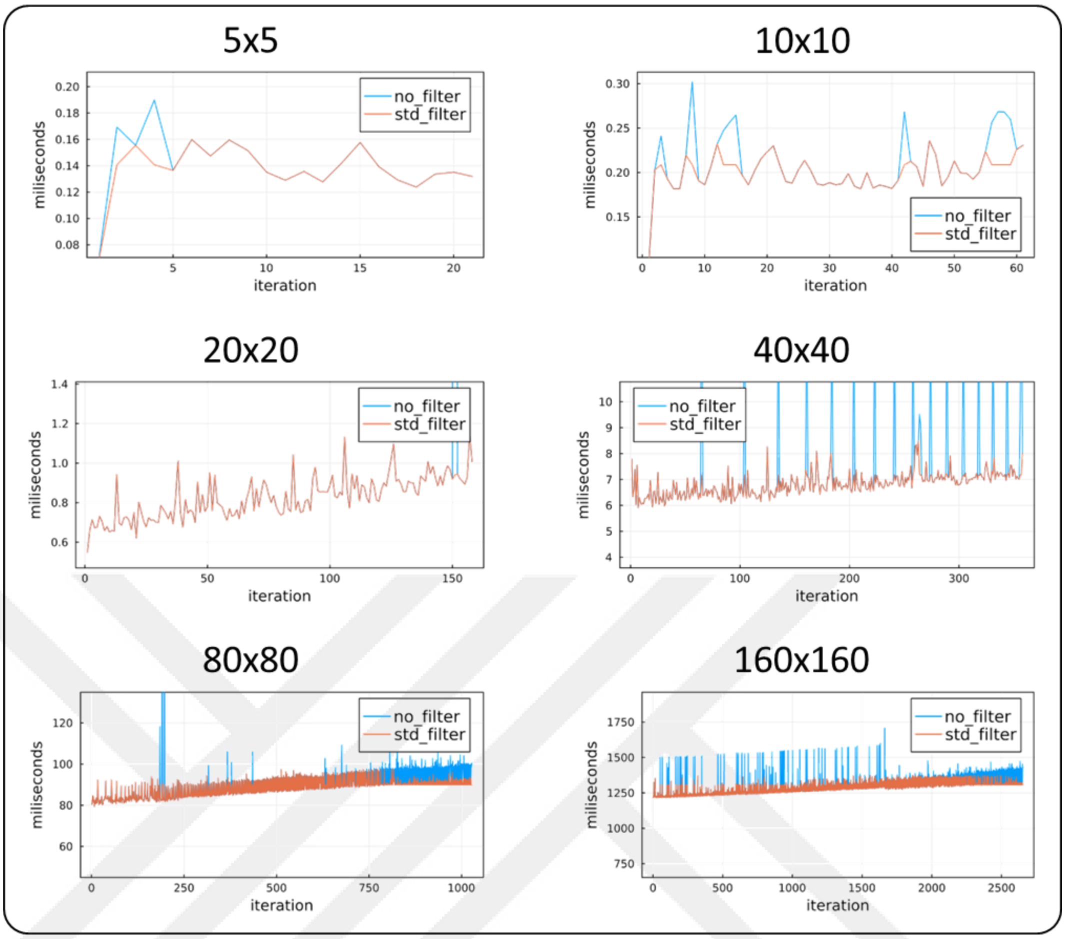


Figure 4.34. *Elapsed time per iterations for different grid resolutions*

Mean value of filtered values of elapsed time per iterations corresponds to reference work unit as defined. Even though elapsed time values had a slightly increasing trend while having unpredictable spikes and variations as seen in Figure 4.34, average of the values was convenient to use as a reference to non-dimensionally compare multigrid cycle performances. Thus, reference work unit variation with respect to grid resolution was explored and measurement metrics were given in Table 4.3 alongside domain information.

Table 4.3. *Reference work unit measurements for different grid resolutions*

Number of Divisions	Number of Cells	Reference Work Unit	Number of Divisions	Number of Cells	Reference Work Unit
5 x 5	25	0.146 ms	100 x 100	10000	190.6 ms
10 x 10	100	0.180 ms	120 x 120	14400	367.5 ms
20 x 20	400	0.658 ms	140 x 140	19600	679.1 ms
30 x 30	900	2.24 ms	160 x 160	25600	1151 ms
40 x 40	1600	6.27 ms	180 x 180	32400	1831 ms
50 x 50	2500	13.32 ms	200 x 200	40000	2789 ms
60 x 60	3600	26.74 ms	250 x 250	62500	6720 ms
70 x 70	4900	47.81 ms	300 x 300	90000	13966 ms
80 x 80	6400	80.1 ms	350 x 350	122500	25822 ms
90 x 90	8100	124.0 ms	400 x 400	160000	44154 ms

Reference work unit curve with respect to number of cells was given in Figure 4.35 together with fitted polynomial curve and was illustrated with logarithmic axes. Reference work unit relation with respect to number of cells was given in Equation 4.4 with almost perfect polynomial fit.

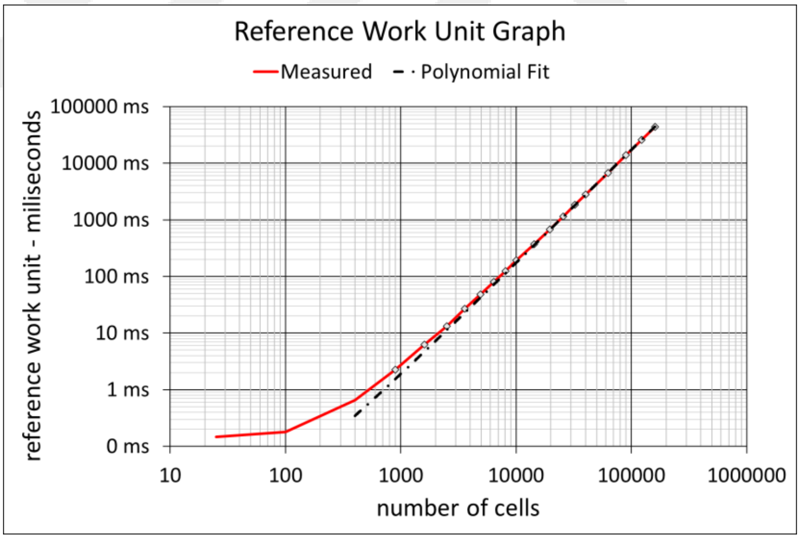


Figure 4.35. *Reference work unit curve as a function of number of cells*

$$RWU = 1.72\,x\,10^{-6}\,x\,(NoC)^2 + 1.76\,x\,10^{-4}\,x\,(NoC) \quad R^2 \cong 1 \qquad (4.4)$$

Curve given in Figure 4.35 and relation given in Equation 4.4 were shared only to prove existence of such relation and/or behaviour. Numerous dependencies and nuisance factors prevented the extraction of a universal relation regarding reference work unit. Figure 4.35 and Equation 4.4 was only applicable to the computer, operating system and environment that studies were conducted. It should also be noted that below certain number of cells, reference work unit did not improve as expected because machine and/or procedures did not respond more faster than a certain rate.

Reference work unit for multigrid cycles should be calculated by division of sum of elapsed time on finest grid iterations to sum of iteration counts on finest grid which in fact returns the average of available values.

Since time cost was selected as the performance indicator of multigrid cycles, rate of change in mean residual with respect to time was also inspected to reveal if its conceptually useful to monitor solution progress. Thus, rate of changes in mean residual with respect to time and iteration were plotted and were given in Figure 4.36 for a grid resolution. Example plot to first derivatives of mean residual progress revealed that using time to monitor rate of change was not applicable due to oscillations alongside unpredictable spikes, while using iteration to monitor rate of change was applicable with a smooth curve.

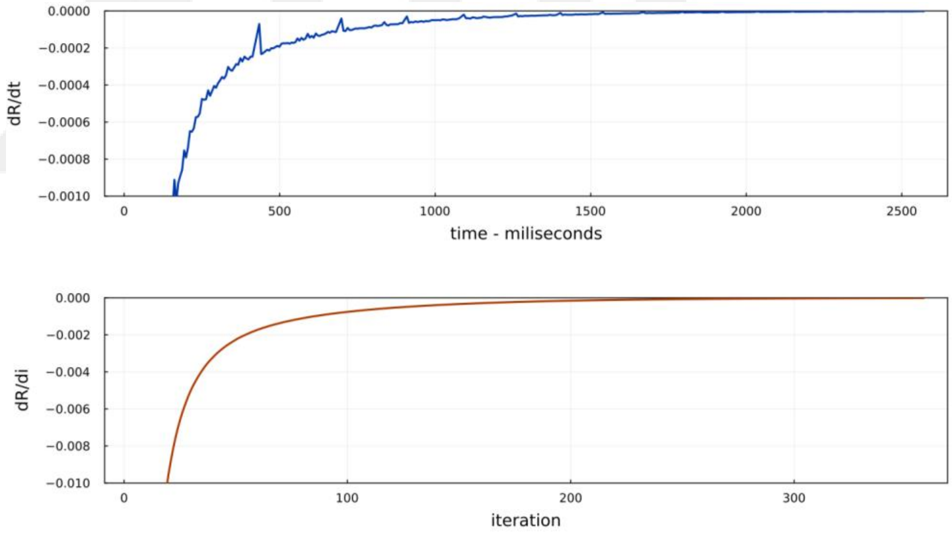


Figure 4.36. *Example to first derivatives of mean residual curves; with respect to time (top) and iteration (bottom)*

Thus, time dependent derivatives were found not usable for monitoring steering criteria or any condition. Even if its plausible at best to sort out the root cause of oscillations and spikes, it was suspected that operating system sub-processes and environment subroutines were mainly responsible. Oscillations and spikes had decreasing effect on fine grid resolutions because of the simple fact that time cost ratio of oscillations and/or spikes to iterations decreased. In any case, time dependent derivatives were not reliable enough to monitor any condition related to solution progress.

4.3.2. Boundary Conditions

Boundary conditions are one of the essential inputs for a finite volume method problem. It is imperative to show that methods and procedures generate output changing with varying input. Method and procedure validity on various boundary conditions should be explored before venturing further. Additionally, validation aside; varying boundary conditions may induce different effects on solution as well as progress. Thus, explorations around changing boundary conditions were conducted to understand implications.

Code accepts Dirichlet and Neumann boundary conditions for the thermal diffusion case. Dirichlet boundary conditions were given as temperature values in the unit of Kelvin. Neumann boundary conditions were given as heat flux values in the unit of Watts.

Randomized boundary condition values were selected from arrays of; 200, 300, 400, 500 °K for Dirichlet and -2000, -1500, -1000, 500, 0, 500, 1000, 1500, 2000 W for Neumann; to each West, East, South and North boundaries. Numerous solutions were generated with the runs using randomized boundary conditions in order to check if methods and/or procedures fail at some. No failures were returned for over hundred cases which validated that methods and procedures work with various boundary conditions. Examples of field contours to such solutions were given in Figure 4.37 and Figure 4.38 to emphasize that methods and procedures were capable of generating solutions.

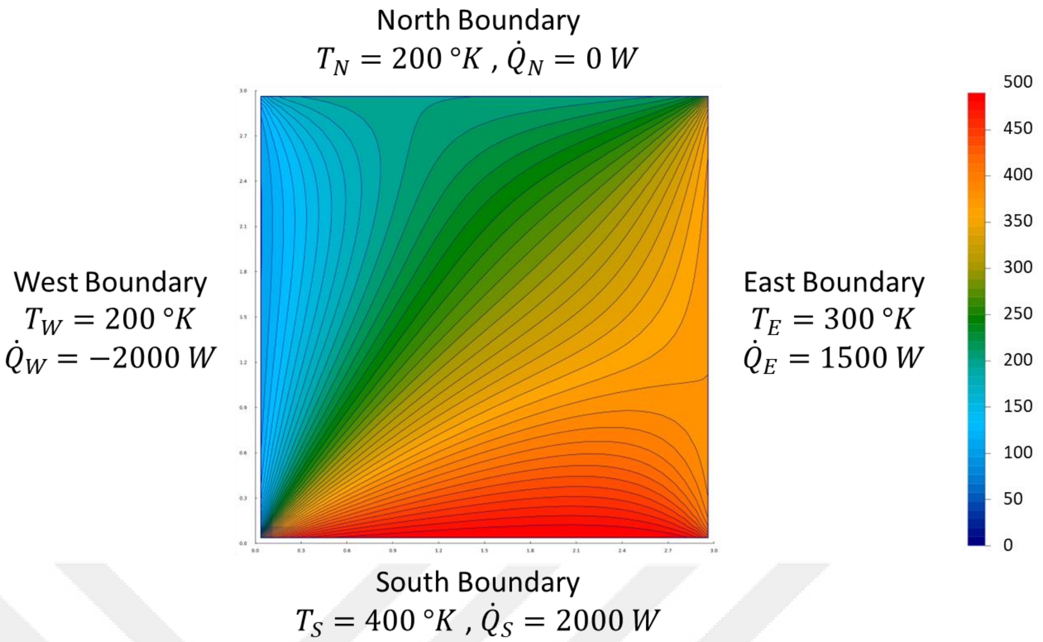


Figure 4.37. *Example #1 of field contour plot to randomized boundary condition cases*

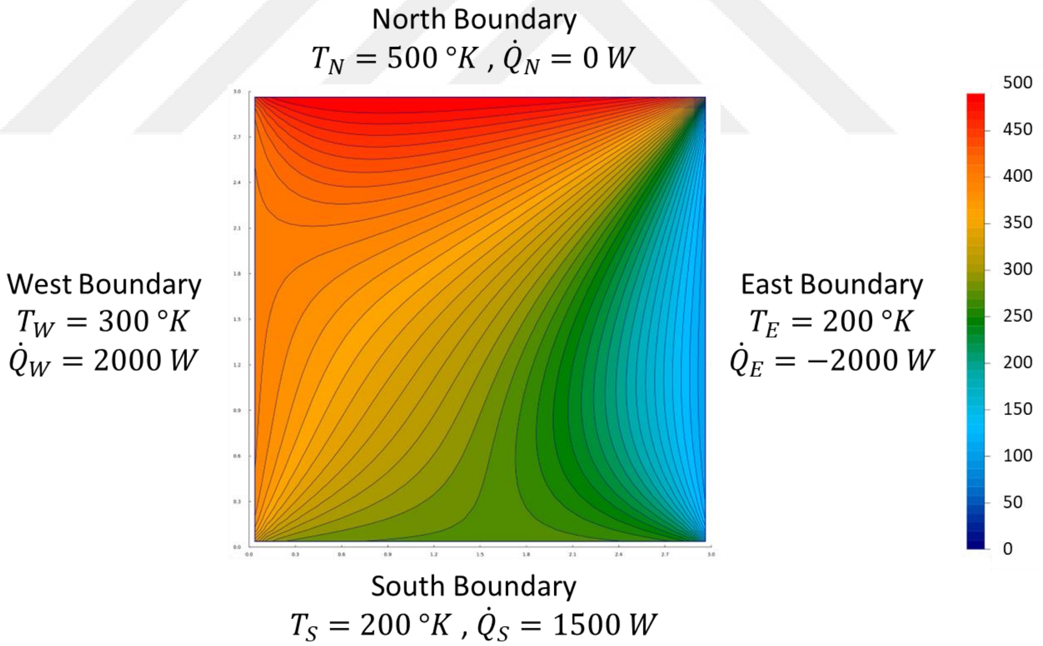


Figure 4.38. *Example #2 of field contour plot to randomized boundary condition cases*

Even though generation of solutions were clarified, any possible effect of boundary conditions on solution progress and/or cost should be explored in an organized manner. Explorations were designed around the reference case changing an individual Dirichlet or Neumann boundary condition at a time for a selected boundary. Grid resolutions also varied within the exploration while convergence values of mean residual were 0.001 for all solutions. Initialization values of domains were calculated by averaging Dirichlet boundary conditions of each case.

Metrics of the exploration that varied Dirichlet conditions to North boundary were given in Table 4.4 and Table 4.5 for inspection. Response surface and contour plot of the Dirichlet boundary condition exploration were given in Figure 4.38 to visualise the metrics. Metrics of the exploration that varied Neumann conditions to North boundary were given in Table 4.6 and Table 4.7 for inspection. Response surface and contour plot of the Dirichlet boundary condition exploration were given in Figure 4.40 to visualise the metrics. Field contour plots of solutions with varying Dirichlet and Neumann boundary conditions were given in Figure 4.37 and Figure 4.39 for the grid resolution of the reference case.

Table 4.4. *Exploration metrics of varying Dirichlet Boundary conditions regarding iteration count*

Iteration Count		North Boundary Temperature Value				
		200 °K	250 °K	300 °K	350 °K	373.15 °K
Grid Resolution	5x5	21	18	18	21	21
	10x10	58	46	48	58	61
	20x20	145	97	102	147	158
	40x40	335	256	266	338	358
	80x80	947	648	686	960	1028

Table 4.5. *Exploration metrics of varying Dirichlet Boundary conditions regarding differences in mean temperature values between initialization and solution*

Absolute Differences in Mean Temperature		North Boundary Temperature Value				
		200 °K	250 °K	300 °K	350 °K	373.15 °K
Grid Resolution	5x5	0.001	0.001	0.002	0.001	0.002
	10x10	0.009	0.009	0.009	0.009	0.009
	20x20	0.040	0.040	0.041	0.040	0.039
	40x40	0.088	0.043	0.047	0.091	0.105
	80x80	0.076	0.034	0.038	0.078	0.093

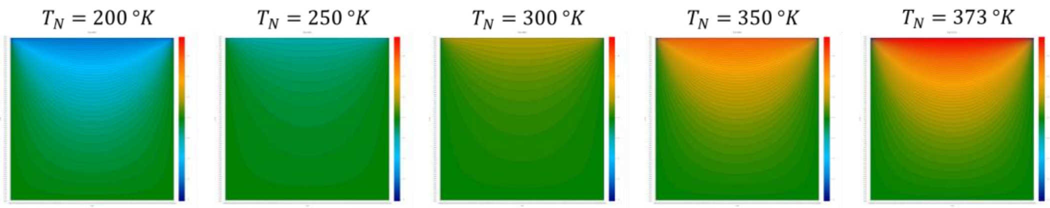


Figure 4.39. *Field contour plots of solutions with varying Dirichlet boundary conditions on North boundary for the grid resolution of the reference case*

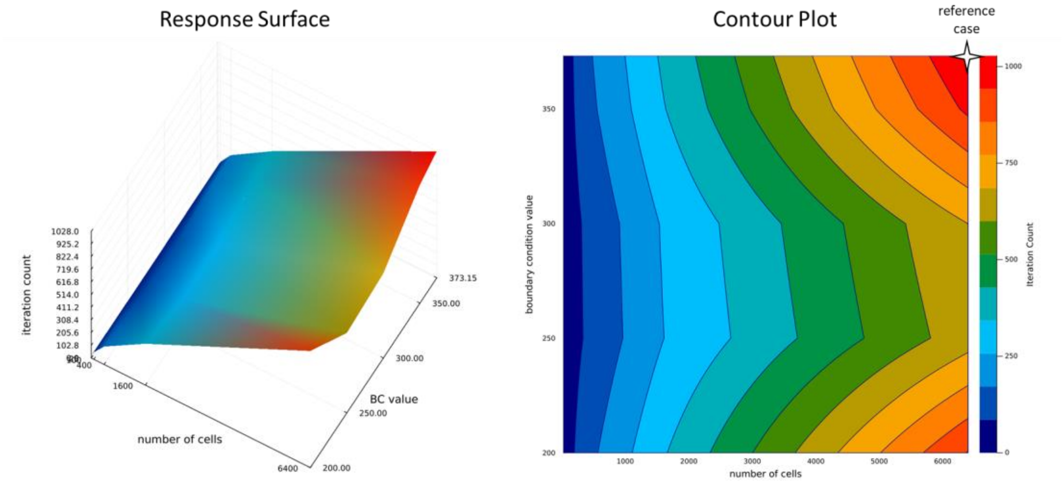


Figure 4.40. *Response surface (left) and contour plot (right) for exploration of Dirichlet boundary condition on North boundary*

Table 4.6. *Exploration metrics of varying Neumann Boundary conditions regarding iteration count*

Iteration Count		West Boundary Heat Flux Value				
		-1000 W	-500 W	0 W	500 W	1000 W
Grid Resolution	5x5	17	20	21	22	23
	10x10	58	39	61	67	71
	20x20	199	154	158	201	221
	40x40	606	470	358	552	647
	80x80	1590	1143	1028	1248	1667

Table 4.7. *Exploration metrics of varying Neumann Boundary conditions regarding differences in mean temperature values between initialization and solution*

Absolute Differences in Mean Temperature		West Boundary Heat Flux Value				
		-1000 W	-500 W	0 W	500 W	1000 W
Grid Resolution	5x5	12.502	6.252	0.002	6.248	12.498
	10x10	12.491	6.259	0.009	6.241	12.491
	20x20	12.460	6.211	0.039	6.211	12.460
	40x40	12.339	6.090	0.105	6.089	12.339
	80x80	11.855	5.644	0.093	5.598	11.852

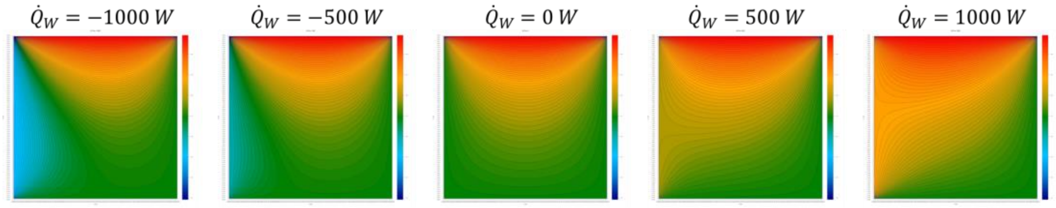
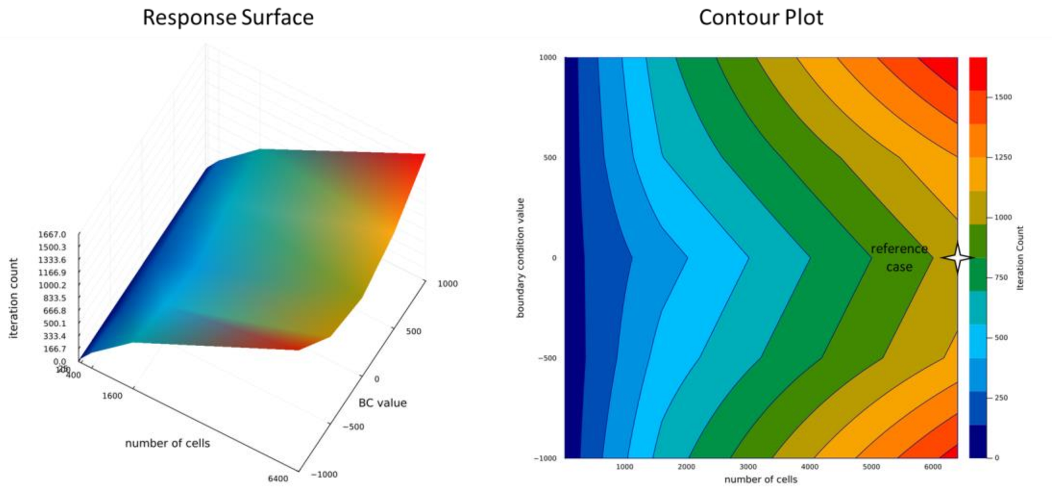


Figure 4.41. *Field contour plots of solutions with varying Neumann boundary conditions on North boundary for the grid resolution of the reference case*



Response Surface

Contour Plot

Figure 4.42. *Response surface (left) and contour plot (right) for exploration of Neumann boundary condition on West boundary*

Temperature values of 200, 250, 300, 350, 373.15 °K were selected around mean temperature value of exact solution which is 298.15 °K for reference case. Heat flux values of -1000, -500, 0, 500, 1000 W were selected to align exploration with conductivity and other inputs of the reference case. Grid resolutions of 5x5, 10x10, 20x20, 40x40, 80x80 were selected to align exploration with other results.

Exploration results were presented with iteration counts as seen in in Table 4.4 and Table 4.6 rather than iteration durations which in fact are the costs of the solutions. Reasoning behind choosing iteration count over iteration duration was to make it easy to read. Solution time with respect to grid resolution changed exponentially as seen in Table 4.2 which would not be practical to use to present exploration results. Additionally, iteration counts of direct iterative solutions were actually cost of the runs in terms of reference work unit at corresponding grid resolution.

Exploration metrics presented in Table 4.5 and Table 4.7 were the absolute of the differences between initialization (temperature) values of the domain and mean (temperature) values of solutions. Initialization values of domains were calculated by averaging Dirichlet boundary conditions of each case.

Inspecting exploration results of Dirichlet boundary condition on North boundary revealed that boundary value itself does not change cost while property distribution of the solution changes cost. In other words, boundary conditions that impose high gradients on domain which in turn decreased convergence rate ending up with increased solution cost.

Thus, main contributor to cost was actually seemed as high gradients rather than boundary conditions.

Additionally, initialization by averaging Dirichlet boundary conditions was seemed appropriate when compared to mean values of solution. Mean values of solutions were almost same with initialization values as seen in Table 4.5 when there were no Neumann boundary conditions present.

Inspecting exploration results of Neumann boundary condition on West boundary revealed that boundary flux changed the complexity and/or gradient within the domain. When the absolute value of flux at boundary increased, convergence rate decreased that ended up with increased solution cost. Increasing absolute values of flux at boundary also increased the absolute difference between initialization and solution.

In both explorations of Dirichlet and Neumann boundary conditions, dominant factor on solution costs were by far grid resolutions. It was safe to assume that boundary conditions are less effective on cost than grid resolution. However, for a certain grid resolution, it seemed that increasing difference between initialization (which was derived from boundary conditions for given explorations) and solution, increased cost as expected. Initialization effect on solution cost increased with increasing grid resolution, which is a factor that should be explored separately.

4.3.3. Initialization Values

Initial guess of the solution is one of the key steps for any computational fluid dynamics problem. Simply, if iterative solution of a problem starts with an initial guess close to solution, iterative process would relatively cost less. Even though initialization effect on solution progress is straightforward enough to understand easily, it is appropriate to quantify the effect and its interaction with grid resolution.

Mean temperature values of 250, 275, 300, 325, 350 °K were used in the initial exploration of the initialization effect, using the reference case values for other parameters. Changing iteration count values with respect to initialization values were given in Figure 4.43 for the reference case. Mean temperature values of solutions with respect to initialization values were given in Figure 4.44 for the reference case.

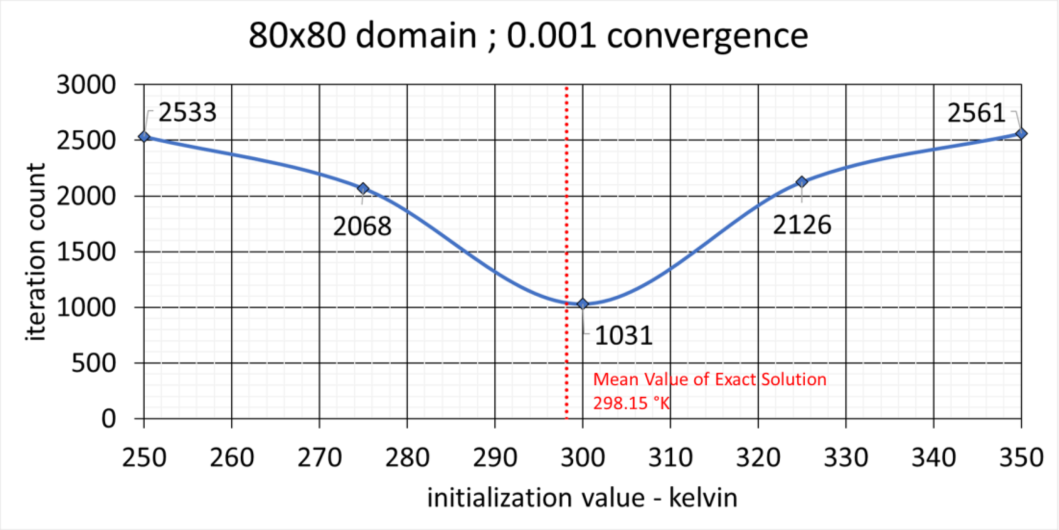


Figure 4.43. *Exploration of iteration count values with respect to initialization values for the reference case*

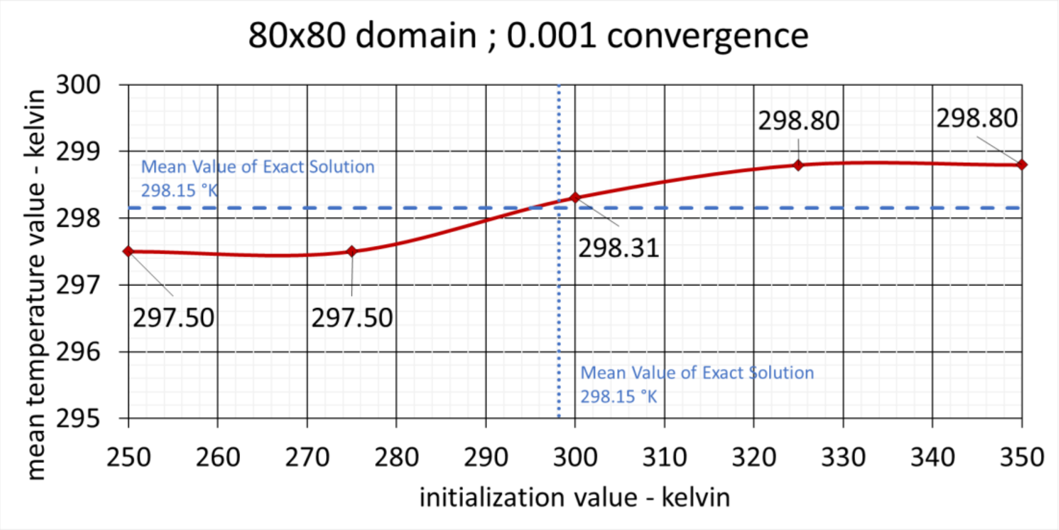


Figure 4.44. *Exploration of mean temperature values of solutions with respect to initialization values for the reference case*

Iteration count values were greatly reduced as seen in Figure 4.43 for initial guess values with increasing proximity to mean value of the solution. Vice versa, solution cost increased with poor initial guess values. Additionally, solution convergence approach to exact solution was directly linked with initial guess values as seen in Figure 4.44 for the reference case. Solution of the initial guesses with higher values than exact solution approached convergence from above, while solution of the initial guesses with lower values than exact solution approached convergence from below.

Exploration of initialization effect was expanded with the addition of grid resolution effect. Convergence values of mean residual were 0.001 for all solutions. Metrics of the exploration that varied initialization values for various grid resolutions were given in Table 4.8 and Table 4.9 for inspection. Response surface and contour plot of the Dirichlet boundary condition exploration were given in Figure 4.45 to visualise the metrics.

Table 4.8. *Exploration metrics of varying initialization values regarding iteration count*

Iteration Count		Initialization Value (°K)				
		250 °K	275 °K	300 °K	325 °K	350 °K
Grid Resolution	5x5	26	24	21	23	25
	10x10	84	78	57	75	82
	20x20	274	246	118	243	272
	40x40	861	748	350	753	864
	80x80	2533	2068	1031	2126	2561

Table 4.9. *Exploration metrics of varying initialization values regarding differences in mean temperature values between initialization and solution*

Absolute Differences in Mean Temperature		Initialization Value (°K)				
		250 °K	275 °K	300 °K	325 °K	350 °K
Grid Resolution	5x5	48.148	23.148	1.852	26.848	51.848
	10x10	48.141	23.141	1.859	26.841	51.841
	20x20	48.111	23.110	1.888	26.811	51.811
	40x40	47.989	22.989	1.819	26.690	51.689
	80x80	47.503	22.502	1.695	26.203	51.203

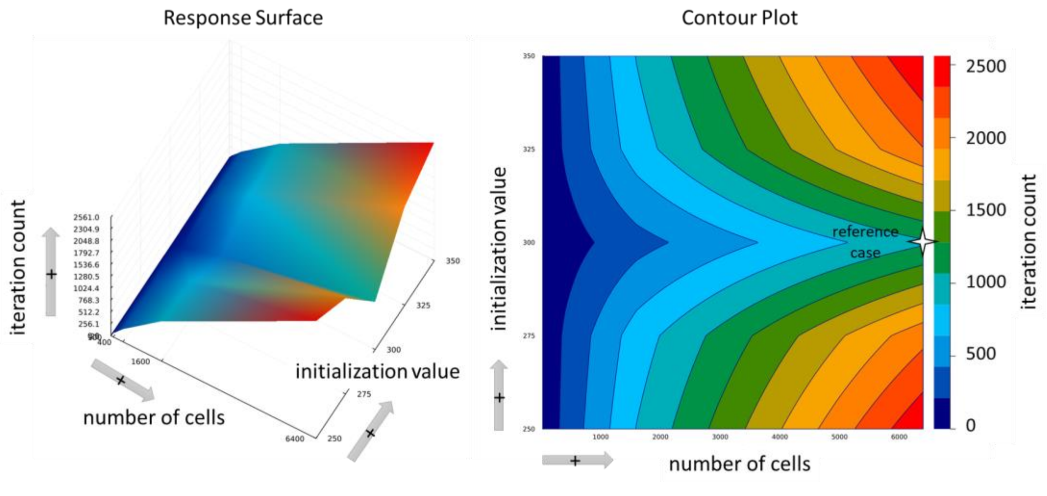


Figure 4.45. *Response surface (left) and contour plot (right) for exploration of initialization values*

Exploration of initialization effect confirmed expectations. With accurate initial guess values, iterative process cost was greatly reduced. Exploration revealed that, initialization effect on solution cost was much higher for fine grid resolutions. As a bonus, exploration also revealed that convergence approach direction towards solution may be selected by picking an appropriate initial guess value. Additionally, it was also confirmed that averaging Dirichlet boundary conditions to calculate initial guess value was a valid approach for the reference case. In essence, best initial guess value would be the value with the most probability density within the domain.

4.3.4. Parameters of Fixed Cycles

Multigrid cycles with fixed schemes were explored around maximum coarse grid level and iterations per sweep. Restriction beyond maximum coarse grid level was infeasible and/or was limited. Iterations per sweep value was the iteration count of solution on each step of multigrid cycle. Exploration was designed to generate cost and convergence values with varying maximum coarse grid level and iteration per sweep. Each multigrid cycle with fixed scheme (V, W, F) was explored separately.

Maximum coarse grid levels were selected as 2, 3, 4, 5 to capture the effect on cost and convergence. Initial grid resolution was selected as 160x160, so minimum grid resolutions are 40x40, 20x20, 10x10, 5x5 respectively. Iterations per sweep values were selected as 1, 2, 3, 4, 5 to capture parameters for minimal cost at acceptable convergence.

Metrics of the exploration of V cycle that varied maximum coarse grid level and iterations per sweep were given in Table 4.10 and Table 4.11 for cost and convergence respectively. Response surface and contour plot of the exploration of V cycle were given in Figure 4.44 and Figure 4.45 to visualise exploration metrics for cost and convergence respectively.

Table 4.10. *Exploration metrics of V cycle with varying maximum coarse level and iterations per sweep regarding cost in reference work unit*

Cost in Reference Work Unit				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	2.996	5.119	7.241	9.512	11.530
	3		20x20	2.974	5.110	7.246	9.405	11.510
	4		10x10	2.959	5.137	7.231	9.397	11.541
	5		5x5	2.979	5.116	7.253	9.410	11.540

Table 4.11. *Exploration metrics of V cycle with varying maximum coarse level and iterations per sweep regarding convergence level of mean residual*

Convergence Level of Mean Residual				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	*2.45719*	0.04608	0.03634	0.03056	0.02663
	3		20x20	*3.55250*	0.01975	0.01492	0.01204	0.01007
	4		10x10	*5.52550*	0.00703	0.00448	0.00300	0.00205
	5		5x5	*6.83194*	0.00137	0.00070	0.00048	0.00038

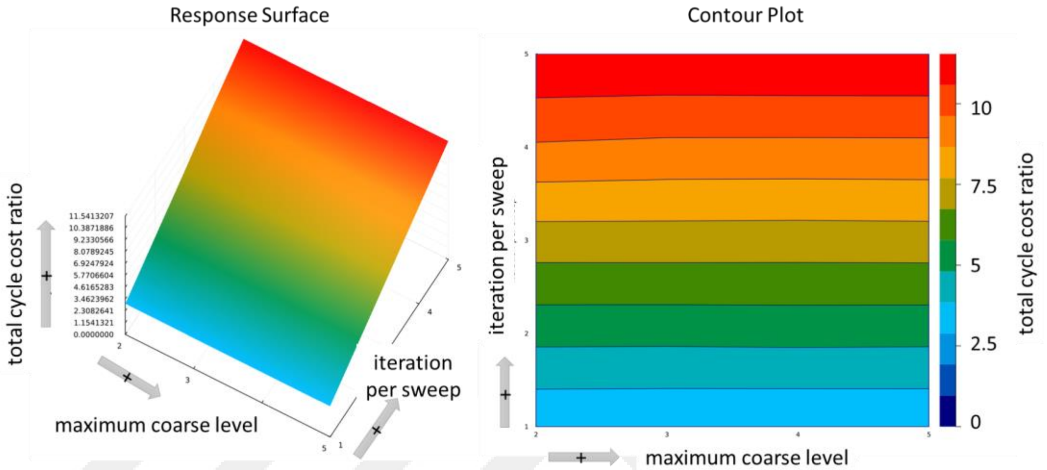


Figure 4.46. *Response surface (left) and contour plot (right) of cost in reference work unit for exploration of V cycle*

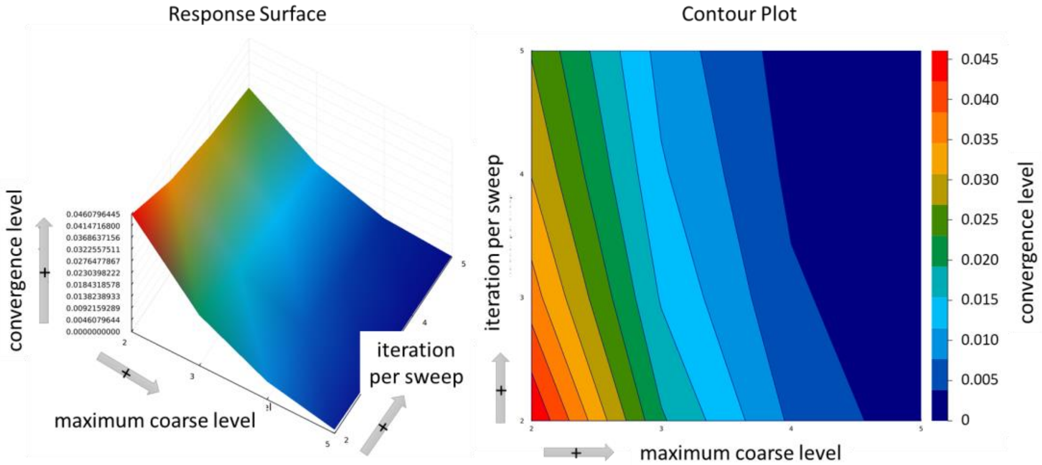


Figure 4.47. *Response surface (left) and contour plot (right) of convergence of mean residual for exploration of V cycle*

Exploration of W cycle was conducted with same input parameters. Metrics of the exploration of W cycle were given in Table 4.10 and Table 4.11 for cost and convergence respectively. Response surface and contour plot of the exploration of W cycle were given in Figure 4.44 and Figure 4.45 to visualise exploration metrics for cost and convergence respectively.

Table 4.12. *Exploration metrics of W cycle with varying maximum coarse level and iterations per sweep regarding cost in reference work unit*

Cost in Reference Work Unit				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	3.523	5.593	7.953	10.334	12.671
	3		20x20	3.083	5.320	7.582	9.776	12.050
	4		10x10	3.083	5.300	7.523	9.770	11.972
	5		5x5	3.091	5.275	7.544	9.742	11.990

Table 4.13. *Exploration metrics of W cycle with varying maximum coarse level and iterations per sweep regarding convergence level of mean residual*

Convergence Level of Mean Residual				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	*2.88776*	0.02155	0.01660	0.01359	0.01152
	3		20x20	*4.24468*	0.00748	0.00494	0.00342	0.00242
	4		10x10	*5.27066*	0.00138	0.00068	0.00046	0.00036
	5		5x5	*3.41220*	0.00091	0.00064	0.00048	0.00039

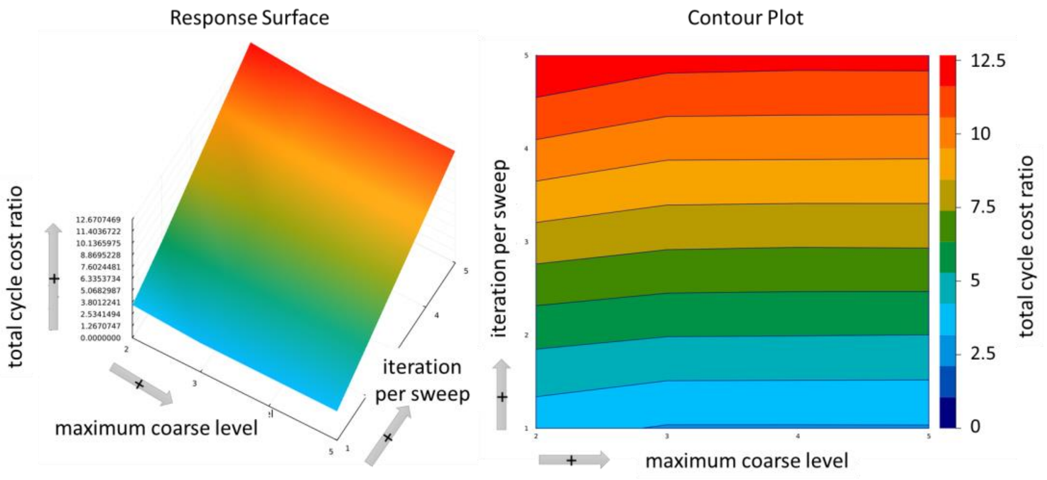


Figure 4.48. *Response surface (left) and contour plot (right) of cost in reference work unit for exploration of W cycle*

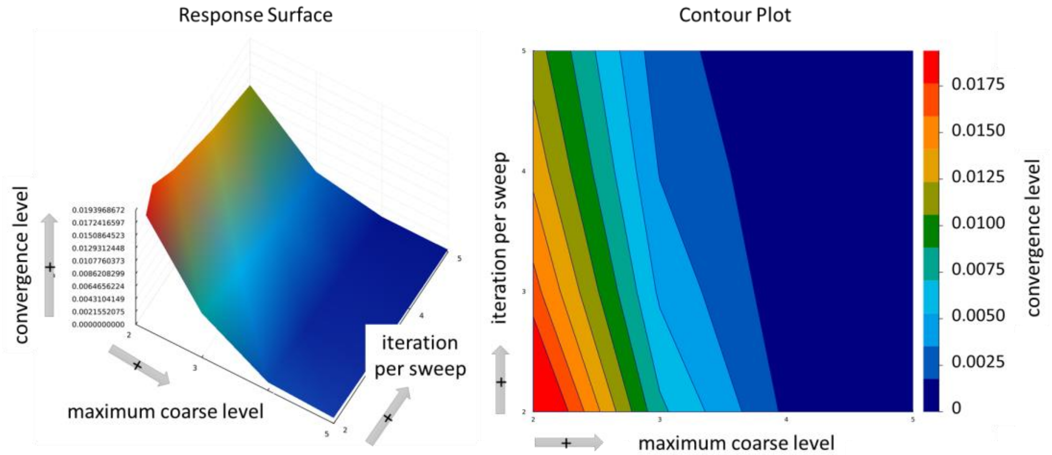


Figure 4.49. *Response surface (left) and contour plot (right) of convergence of mean residual for exploration of W cycle*

Exploration of F cycle was conducted with same input parameters. Metrics of the exploration of F cycle were given in Table 4.10 and Table 4.11 for cost and convergence respectively. Response surface and contour plot of the exploration of F cycle were given in Figure 4.44 and Figure 4.45 to visualise exploration metrics for cost and convergence respectively.

Table 4.14. *Exploration metrics of F cycle with varying maximum coarse level and iterations per sweep regarding cost in reference work unit*

Cost in Reference Work Unit				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	3.062	5.260	7.485	9.668	11.885
	3		20x20	3.082	5.339	7.541	9.773	12.012
	4		10x10	3.075	5.325	7.562	9.789	11.999
	5		5x5	3.106	5.337	7.563	9.781	12.028

Table 4.15. *Exploration metrics of F cycle with varying maximum coarse level and iterations per sweep regarding convergence level of mean residual*

Convergence Level of Mean Residual				Iterations per Sweep				
				1	2	3	4	5
Maximum Coarse Level	2	Minimum Grid Resolution	40x40	*2.55164*	0.03227	0.02530	0.02110	0.01823
	3		20x20	*4.27735*	0.00930	0.00644	0.00470	0.00352
	4		10x10	*5.28459*	0.00128	0.00065	0.00045	0.00036
	5		5x5	*2.68124*	0.00092	0.00064	0.00048	0.00039

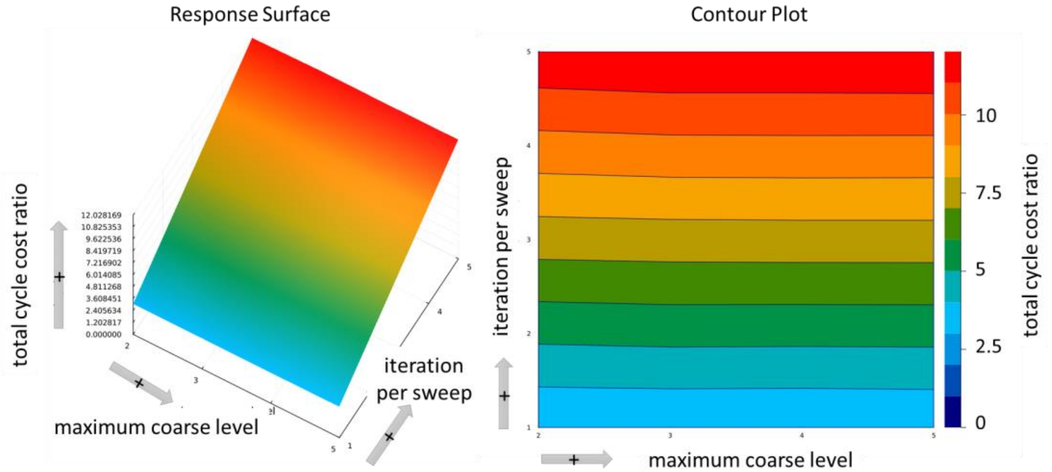


Figure 4.50. *Response surface (left) and contour plot (right) of cost in reference work unit for exploration of F cycle*

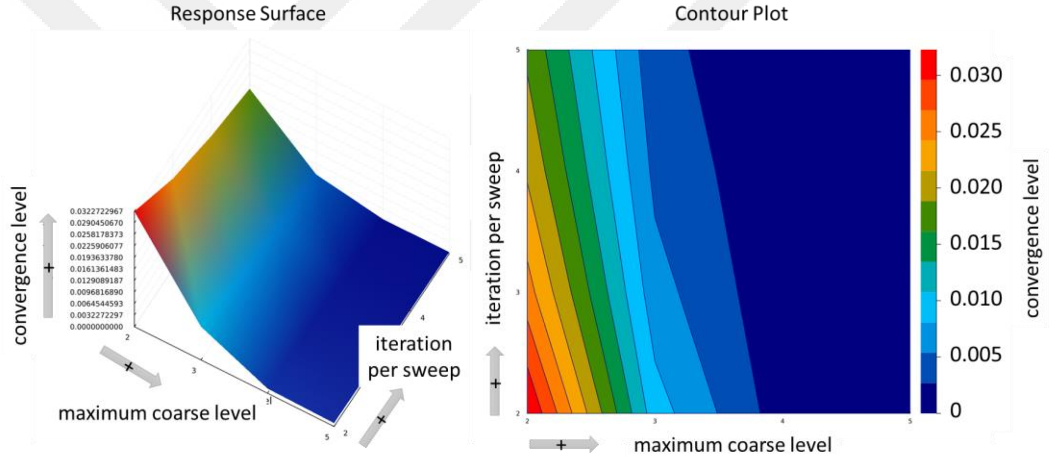


Figure 4.51. *Response surface (left) and contour plot (right) of convergence of mean residual for exploration of F cycle*

Even though numbers do slightly change, explorations of V, W and F multigrid cycles resulted with same inferences.

Single iterations per sweep generated unacceptable solutions as seen in Table 4.11, Table 4.13 and Table 4.15 regarding convergence levels of mean residual. Results of single iterations per sweep was also not comparable with other results of iterations per sweep.

Cost in reference work unit was directly proportional to iterations per sweep while maximum coarse level had little or no effect on cost. Convergence level of mean residual improved with increasing maximum coarse grid level. Although iterations per sweep had an effect on convergence, maximum coarse level was dominant factor relatively on convergence.

In order to reduce cost while considering the limitation to minimum number of iterations per sweep, any multigrid cycle should visit maximum feasible coarse grid resolution level. In this manner, a reference multigrid cycle can be defined with 2 (two) iterations per sweep and maximum coarsening possible based on V scheme. Any multigrid cycle simply can not have lower cost than the reference multigrid cycle. Thus, the reference multigrid cycle can be used to evaluate performance of multigrid cycles with adaptive algorithms.

Run summary plot of the reference multigrid cycle was given in Figure 4.52 to illustrate the shortest possible path to solution for the reference case at 160x160 grid resolution. Solution convergence level of the reference multigrid cycle was 0.001373 of mean residual. Total cost of the reference multigrid cycle can be assumed as 5 (five) reference work units. Consistency of the cost of the reference multigrid cycle was also investigated. Distribution of reference work unit measurements of reference multigrid cycle with 100 samples were plotted and was given in Figure 4.53 accompanied by statistical properties. According to measurements presented in Figure 4.53 reference work unit may variate 7.5% around the mean for extremes while having lower and higher quartiles within 1% of mean.

Inspecting cost of each item in Figure 4.52 leaded to observation that iterations cost had the highest percentage of the total cost. Furthermore, iterations on the finest grid resolution had the highest contribution with overwhelming percentage. Cost of iterations made up to 85% of the total cost while finite volume method operations costed 13% of the total costed and intergrid operations costed 2% of the total cost. Cost of iterations on finest grid resolution made up 80% percent of the total cost. It is imperative to remind that these cost fractions were for the best possible case and/or path. Observation of the cost fractions underlined the cost contribution of initialization and finalization steps. Compared to initialization and finalization cost, all operations and iterations to employ multigrid cycles costed extremely low.

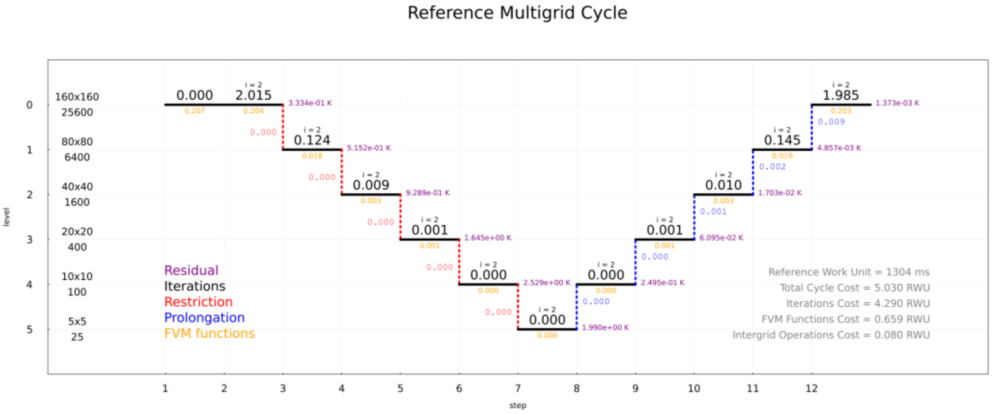


Figure 4.52. *Example to run summary plot to the reference multigrid cycle*

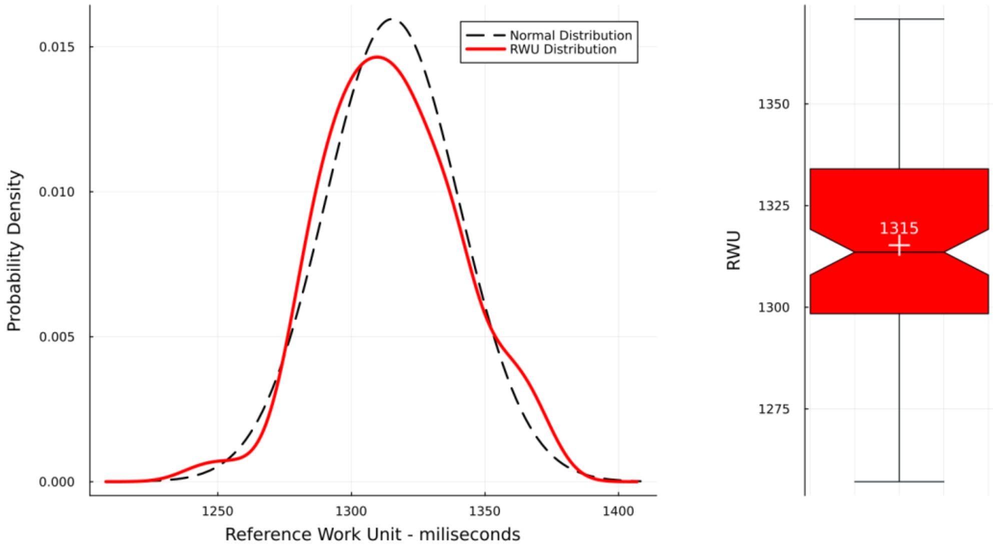


Figure 4.53. *Statistical data of reference work unit measurements of the reference multigrid cycle*

4.3.5. Parameters of Flexible Cycle

Flexible multigrid cycles with adaptive scheme were explored around steering criteria. Alpha parameter was the criteria for prolongation, beta parameter was the criteria for restriction. Alpha was defined as the ratio of convergence level of current iteration to convergence level of previous step that triggers prolongation. Beta was defined as the ratio of convergence levels of consecutive iterations that triggers restriction. Restriction was allowed up to infeasible grid level. Exploration was designed to generate cost and convergence values with varying alpha and beta parameters.

Alpha values were initially selected varying between 0.05 and 0.50 with 0.05 steps; and beta values were initially selected varying between 0.50 and 0.95 with 0.05 steps; to capture the effect on cost and convergence. Default values of alpha and beta were noted as 0.3 and 0.7 respectively [16]. Initial grid resolution was selected as 160x160, so minimum grid resolution was 5x5 at coarsest level. Since the cycle scheme was adaptive, operational limitations were set. Minimum iterations per sweep was 2, maximum iterations per sweep was 50, maximum number of steps was 200 and final convergence target was 0.001 mean residual.

Metrics of the initial exploration of flexible cycle at broad space that varied alpha and beta were given in Table 4.16 and Table 4.17 for cost and convergence respectively. Response surface points and contour view of points of the exploration of flexible cycle were given in Figure 4.54 and Figure 4.55 to visualise exploration metrics for cost and convergence respectively.

Flexible cycles had tendency to stuck in a restriction and prolongation loop at coarse levels for some couples of alpha and beta while solving the reference case. Solutions were only acceptable at the initial resolution level. Thus, invalid runs that were not complying with operational limitations were excluded and were marked. There were 77 valid runs in of broad space with 100 points.

Table 4.16. *Exploration metrics of flexible cycle with varying alpha and beta regarding cost in reference work unit at broad space*

cost		beta									
		0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
alpha	0.05	6.21	6.22	6.20	6.20	6.27	7.20	7.29	8.30	10.43	15.81
	0.10	6.20	6.28	🟥	6.20	🟥	7.19	7.30	8.33	10.45	15.78
	0.15	6.21	6.26	🟥	🟥	🟥	7.21	7.29	8.25	13.57	15.99
	0.20	6.21	6.26	🟥	🟥	🟥	🟥	7.28	11.43	11.61	16.03
	0.25	8.42	8.41	🟥	🟥	🟥	🟥	9.41	9.79	10.96	16.42
	0.30	🟥	8.42	🟥	🟥	!	🟥	9.42	🟥	15.57	21.87
	0.35	7.24	8.19	8.20	8.22	8.19	8.20	8.44	🟥	15.61	24.21
	0.40	7.18	8.20	8.23	8.18	8.25	8.18	8.27	🟥	15.45	22.01
	0.45	9.37	10.41	10.37	10.42	10.36	8.25	🟥	🟥	18.58	21.85
	0.50	9.37	10.36	10.37	10.42	10.37	8.18	13.04	13.06	17.68	21.77

Table 4.17. *Exploration metrics of flexible cycle with varying alpha and beta regarding convergence of mean residual at broad space*

convergence		beta									
		0.50	0.55	0.60	0.65	0.70	0.75	0.80	0.85	0.90	0.95
alpha	0.05	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.10	0.00052	0.00052	🟥	0.00052	🟥	0.00052	0.00052	0.00052	0.00052	0.00052
	0.15	0.00052	0.00052	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052
	0.20	0.00052	0.00052	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052
	0.25	0.00052	0.00052	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052
	0.30	🟥	0.00052	🟥	🟥	!	🟥	0.00052	🟥	0.00052	0.00052
	0.35	0.00094	0.00098	0.00098	0.00098	0.00096	0.00094	0.00052	🟥	0.00052	0.00058
	0.40	0.00094	0.00098	0.00098	0.00098	0.00096	0.00094	0.00074	🟥	0.00052	0.00067
	0.45	0.00085	0.00085	0.00085	0.00085	0.00080	0.00094	🟥	🟥	0.00052	0.00080
	0.50	0.00085	0.00085	0.00085	0.00085	0.00080	0.00099	0.00070	0.00080	0.00052	0.00084

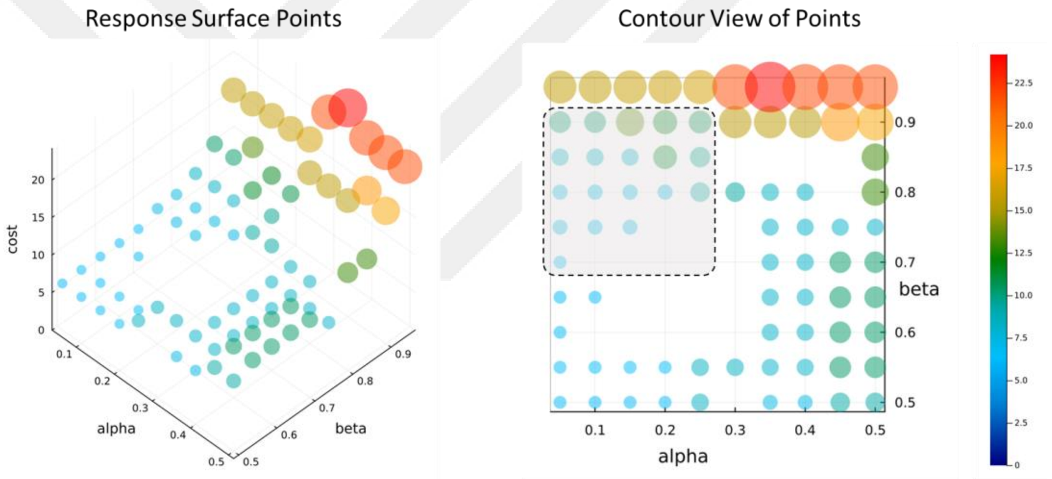
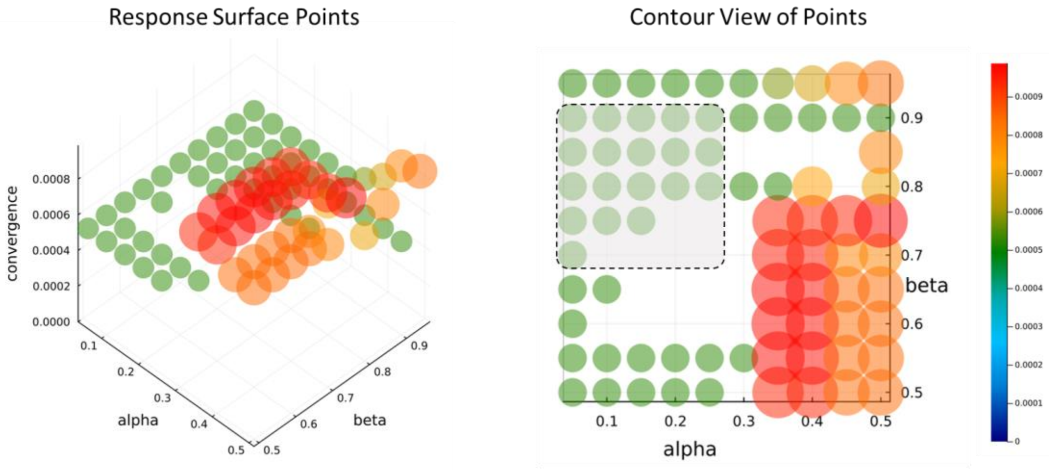


Figure 4.54. *Response surface points (left) and contour view of points (right) of cost in reference work unit for exploration of flexible cycle at broad space*


Response Surface Points

Contour View of Points

Figure 4.55. *Response surface points (left) and contour view of points (right) of convergence of mean residual for exploration of flexible cycle at broad space*

High beta values ended up with relatively high cost as seen in Figure 4.52 while high values of alpha ended up with relatively high convergence levels as seen in Figure 4.53. Most of the moderate values of alpha and beta generated invalid runs. Interestingly enough, default values of alpha and beta did not work with the reference case at initial resolution of exploration.

According to results of initial exploration of flexible cycle at broad space, following exploration at a narrower space was designed as seen in Figure 4.54 and Figure 4.55 as a region encompassed with dashed lines. Alpha value varied 0.05 to 0.25 with 0.01 step and beta value varied 0.70 to 0.90 with 0.01 step for the exploration of the narrow space. Selected space for the following exploration had higher resolution than the initial exploration to uncover precise boundary of validity.

Metrics of the following exploration of flexible cycle at narrow space that varied alpha and beta were given in Table 4.18 and Table 4.19 for cost and convergence respectively. Response surface points and contour view of points of the exploration of flexible cycle were given in Figure 4.56 and Figure 4.57 to visualise exploration metrics for cost and convergence respectively.

Table 4.18. *Exploration metrics of flexible cycle with varying alpha and beta regarding cost in reference work unit at narrow space*

cost		beta																				
		0.70	0.71	0.72	0.73	0.74	0.75	0.76	0.77	0.78	0.79	0.80	0.81	0.82	0.83	0.84	0.85	0.86	0.87	0.88	0.89	0.90
alpha	0.05	6.22	6.32	6.23	6.23	6.31	7.21	7.20	7.23	7.21	7.29	7.29	7.35	7.31	8.29	8.28	8.29	8.37	9.33	9.36	9.46	10.50
	0.06	6.21	6.22	6.29	6.21	6.21	7.29	7.20	7.22	7.30	7.27	7.28	7.40	7.28	8.29	8.42	8.30	8.40	9.45	9.37	9.47	10.51
	0.07	6.20	6.21	6.21	6.21	6.23	7.23	7.22	7.21	7.24	7.36	7.27	7.30	7.42	8.31	8.30	8.30	8.35	9.36	9.44	9.43	10.44
	0.08	🟥	🟥	🟥	🟥	6.24	7.29	7.21	7.21	7.21	7.27	7.29	7.31	7.35	8.28	8.27	8.34	8.34	9.36	9.41	9.41	10.41
	0.09	🟥	🟥	🟥	🟥	🟥	7.21	7.22	7.21	7.20	7.38	7.27	7.29	7.29	8.26	8.28	8.30	8.34	9.34	9.33	9.40	10.41
	0.10	🟥	🟥	🟥	🟥	🟥	7.18	7.21	7.28	7.19	7.28	**7.39**	7.27	7.29	8.27	8.26	8.27	8.34	9.35	9.34	9.50	10.40
	0.11	🟥	🟥	🟥	🟥	🟥	7.28	7.18	7.21	7.28	7.26	7.26	7.38	7.28	8.29	8.35	8.27	8.34	9.42	9.33	9.41	10.46
	0.12	🟥	🟥	🟥	🟥	🟥	🟥	7.22	7.27	7.19	7.27	7.37	7.27	7.28	8.28	8.26	8.28	8.34	9.34	9.33	9.45	10.40
	0.13	6.19	6.19	6.19	6.18	6.20	🟥	🟥	🟥	7.18	7.32	7.25	7.27	7.24	8.26	8.28	8.24	8.33	9.35	9.32	9.40	10.42
	0.14	🟥	🟥	🟥	6.20	6.26	7.19	7.20	7.19	7.18	7.27	7.36	7.27	7.28	8.35	8.26	8.28	8.42	9.33	9.35	9.47	10.41
	0.15	🟥	🟥	🟥	🟥	🟥	7.20	7.20	7.22	7.27	7.28	7.27	7.24	7.26	8.28	8.34	8.25	8.35	9.31	9.34	9.42	13.53
	0.16	🟥	🟥	🟥	🟥	🟥	7.28	7.21	7.20	7.34	7.27	7.29	7.37	7.27	8.28	8.36	8.26	8.34	12.53	12.46	12.57	12.61
	0.17	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	7.28	7.34	7.26	7.28	8.35	8.24	8.26	8.42	12.46	12.48	12.53	12.54
	0.18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	7.34	7.26	7.28	7.35	11.39	11.41	11.50	11.49	11.48	11.47	11.63	11.57
	0.19	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	7.29	7.36	7.26	7.30	11.48	11.42	11.52	11.54	11.56	11.51	11.61	11.67
	0.2	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	7.29	7.30	7.28	7.33	11.46	11.47	11.45	11.61	11.54	11.61	11.62	11.61
	0.21	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	10.43	10.51	10.42	10.48	10.54	10.46	10.58	10.51	10.65	10.57	10.59	10.70
	0.22	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	10.48	10.42	10.45	10.51	10.44	10.58	10.50	10.53	10.69	10.60	10.63
	0.23	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	10.41	🟥	🟥	🟥	10.52	10.51	10.61	10.52	10.59	10.57	10.66
	0.24	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	10.40	🟥	🟥	🟥	10.45	10.52	10.58	10.51	10.68	10.90	10.96
	0.25	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	9.41	🟥	🟥	🟥	9.77	9.82	9.49	9.58	9.69	10.00	11.02

Table 4.19. *Exploration metrics of flexible cycle with varying alpha and beta regarding convergence of mean residual at narrow space*

conv.		beta																				
		0.70	0.71	0.72	0.73	0.74	0.75	0.76	0.77	0.78	0.79	0.80	0.81	0.82	0.83	0.84	0.85	0.86	0.87	0.88	0.89	0.90
alpha	0.05	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.06	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.07	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.08	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.09	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.10	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	**0.00052**	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.11	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.12	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.13	0.00052	0.00052	0.00052	0.00052	0.00052	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.14	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.15	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.16	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.17	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.18	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.19	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.2	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.21	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.22	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.23	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.24	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052
	0.25	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	0.00052	🟥	🟥	🟥	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052	0.00052

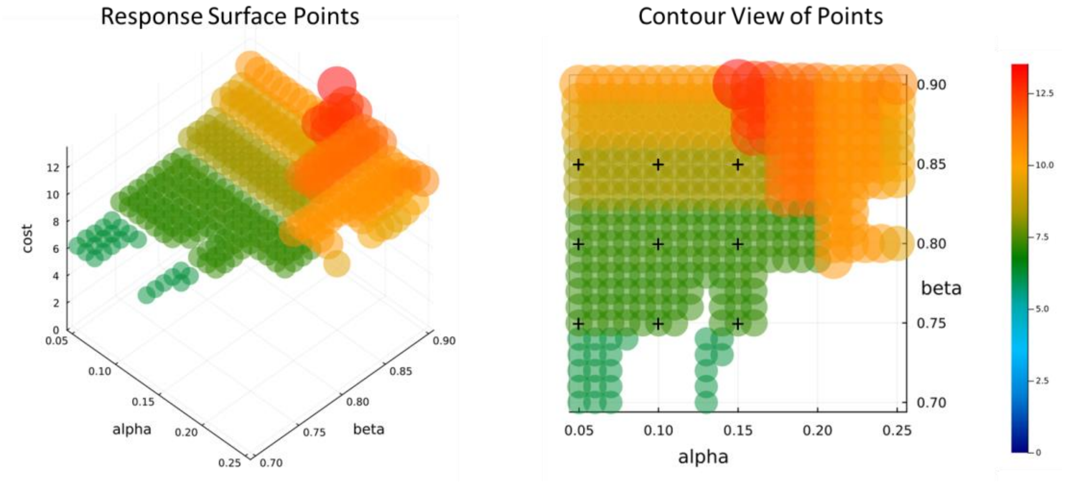


Figure 4.56. *Response surface points (left) and contour view of points (right) of cost in reference work unit for exploration of flexible cycle at narrow space*

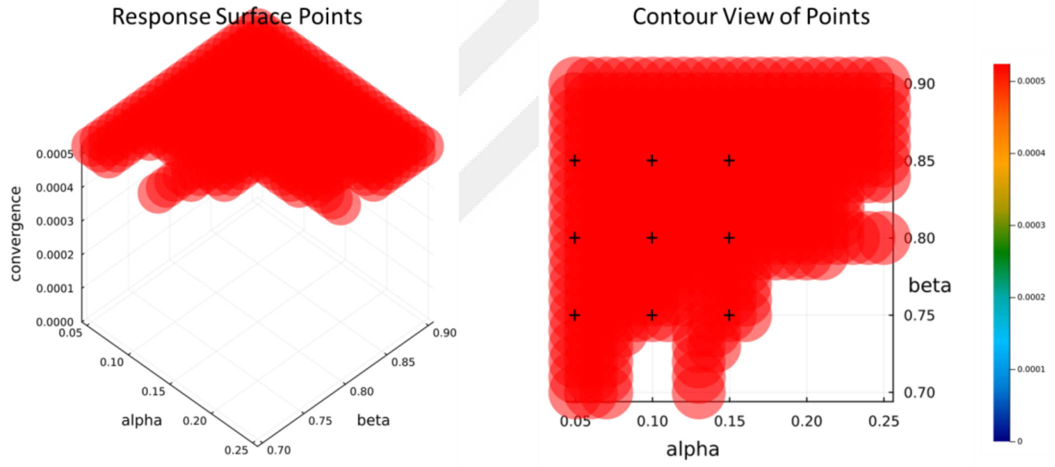


Figure 4.57. *Response surface points (left) and contour view of points (right) of convergence of mean residual for exploration of flexible cycle at narrow space*

Although cost of the flexible multigrid cycle varied irregularly with changing alpha and beta for exploration at narrow space; convergence levels of mean residuals were identical regardless of alpha and beta as seen in Figure 4.56 and Figure 4.57 for the valid runs. Several couples of alpha and beta were selected for further inspection as best performing sample points of exploration space

Detailed cost metrics of flexible multigrid cycle at selected points were given in Table 4.20 for various parameters with average values. Reference work unit (RWU) variation is the percentage difference between RWU measurement of the instance and the mean of the measurements. Iteration cost is the percentage cost of the iterations within the cycle. Finite volume cost is the percentage cost of the finite volume method functions within the cycle. Intergrid cost is the percentage cost of the intergrid operations (restriction, prolongation) within the cycle. Total costs of the cycles were given in Figure 4.58 alongside the performances of the flexible cycle at selected points.

Table 4.20. *Cost metrics of flexible multigrid cycles at selected points*

RWU variation		beta		
		0.75	0.80	0.85
alpha	0.05	0.44%	0.26%	0.62%
	0.10	-0.40%	-0.35%	-0.14%
	0.15	-0.18%	0.14%	-0.39%

iteration cost		beta		
		0.75	0.80	0.85
alpha	0.05	90.2%	90.4%	91.6%
	0.10	90.3%	89.0%	91.6%
	0.15	90.3%	90.4%	91.6%

finite volume cost		beta		
		0.75	0.8	0.85
alpha	0.05	8.7%	8.5%	7.5%
	0.10	8.7%	9.9%	7.5%
	0.15	8.7%	8.5%	7.5%

intergrid cost		beta		
		0.75	0.80	0.85
alpha	0.05	1.1%	1.1%	0.9%
	0.10	1.0%	1.1%	0.9%
	0.15	1.0%	1.1%	0.9%

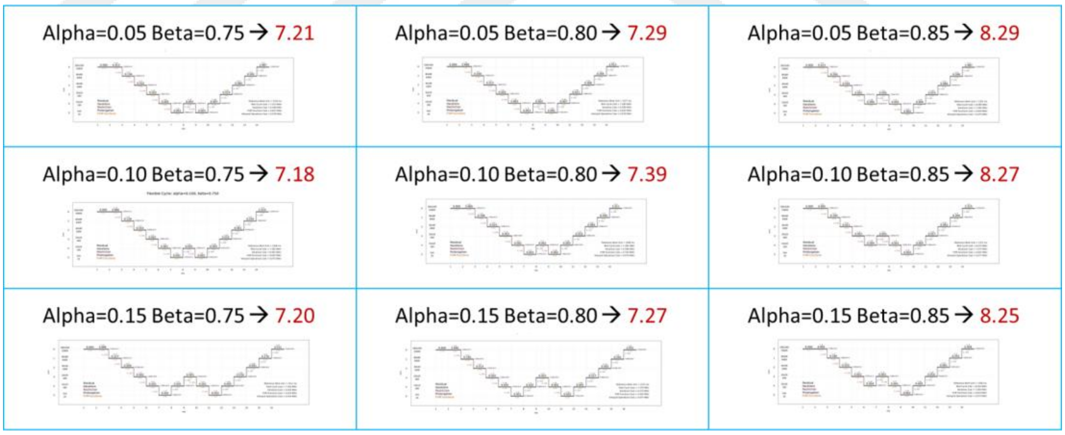


Figure 4.58. *Performances of flexible multigrid cycles at selected points*

Best cases of flexible cycle run resided within the region that has alpha values between 0.05 and 0.25; and beta values between 0.75 and 0.85 for the reference case. Best performing flexible cycles costed 7.60 reference work unit on average while having 0.00052 convergence level of mean residual. Run summary plots of the best cases of the flexible cycle were given in Appendix #6 for further inspection.

4.3.6. Parameters of PID Driven Cycle

PID driven multigrid cycles with adaptive scheme were explored around steering criteria. There were several parameters that contributes to steering criteria. Proportional, integral and derivative coefficients were used to calculate proportional, integral and derivative terms respectively. Conditional checks between respective proportional, integral and derivative terms and criteria triggered corresponding action which may be restriction, prolongation as well as continuation to iterations without changing grid resolution. Restriction was allowed up to infeasible grid level. Exploration was designed to generate cost and convergence values with varying PID coefficients and PID criteria.

Since conditional checks were done against PID criteria, establishing valid and robust criteria was imperative before venturing towards tuning PID coefficients. Proportional criterion (Pc) values were initially selected varying between 0.10 and 0.50 with 0.05 steps. Integral criterion (Ic) values were initially selected varying between 5.0 and 20.0 with 5.0 steps. Derivative criterion (Dc) values were initially selected varying between -0.25 and -0.05 with 0.05 steps. Default values of proportional, integral and derivative coefficients were given as 1.0, 1.0 and 1.0 respectively. Initial grid resolution was selected as 160x160, so minimum grid resolution was 5x5 at coarsest level. Since the cycle scheme was adaptive, operational limitations were set. Minimum iterations per sweep was 2, maximum iterations per sweep was 50, maximum number of steps was 200 and final convergence target was 0.001 mean residual.

Metrics of the initial exploration of PID driven cycle at broad space that varied PID criteria were given in Table 4.21 and Table 4.22 for cost and convergence respectively. Response surface points and contour view of points of the exploration of PID driven cycle were given in Figure 4.59 and Figure 4.60 to visualise exploration metrics for cost and convergence respectively.

PID driven cycles had tendency to stuck in a restriction and prolongation loop at various grid resolution levels for some sets of PID coefficient and criteria while solving the reference case. Solutions are only acceptable at the initial resolution level. Thus, invalid runs that were not complying with operational limitations were excluded and were marked. There were 39 valid runs in of broad space with 45 points.

Table 4.21. *Exploration metrics of PID driven cycle with varying PID criteria regarding cost in reference work unit at broad space (for integral criterion = 5.0)*

Cost (Integral Criterion = 5.0)		Derivative Criterion				
		-0.25	-0.20	-0.15	-0.10	-0.05
Proportional Criterion	0.10	🟥	🟥	🟥	🟥	7.41
	0.15	🟥	🟥	6.25	6.22	7.39
	0.20	6.09	6.08	6.08	6.22	7.47
	0.25	6.01	6.14	6.05	6.14	11.09
	0.30	7.05	7.05	6.06	6.14	10.93
	0.35	6.99	7.06	6.06	9.73	12.05
	0.40	6.99	7.17	6.08	13.43	11.94
	0.45	9.60	9.66	9.66	13.45	15.71
	0.50	9.68	9.68	14.34	13.39	19.17

Table 4.22. *Exploration metrics of PID driven cycle with varying PID criteria regarding convergence of mean residual at broad space (for integral criterion = 5.0)*

Convergence (Integral Criterion = 5.0)		Derivative Criterion				
		-0.25	-0.20	-0.15	-0.10	-0.05
Proportional Criterion	0.10	🟥	🟥	🟥	🟥	0.00060
	0.15	🟥	🟥	0.00063	0.00069	0.00068
	0.20	0.00077	0.00077	0.00077	0.00075	0.00079
	0.25	0.00095	0.00094	0.00091	0.00088	0.00087
	0.30	0.00087	0.00085	0.00098	0.00096	0.00097
	0.35	0.00087	0.00085	0.00098	0.00091	0.00086
	0.40	0.00087	0.00085	0.00098	0.00091	0.00092
	0.45	0.00094	0.00093	0.00093	0.00094	0.00085
	0.50	0.00100	0.00093	0.00087	0.00098	0.00097

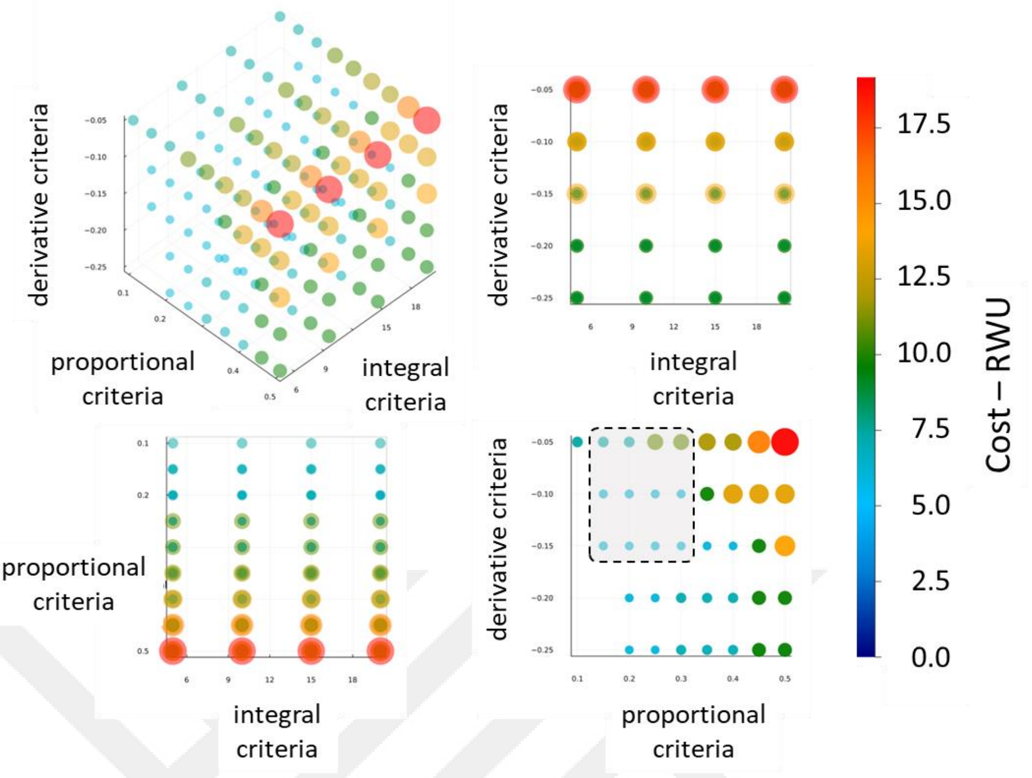


Figure 4.59. *Response points and orthogonal views of cost in reference work unit for exploration of PID driven cycle criteria at broad space*

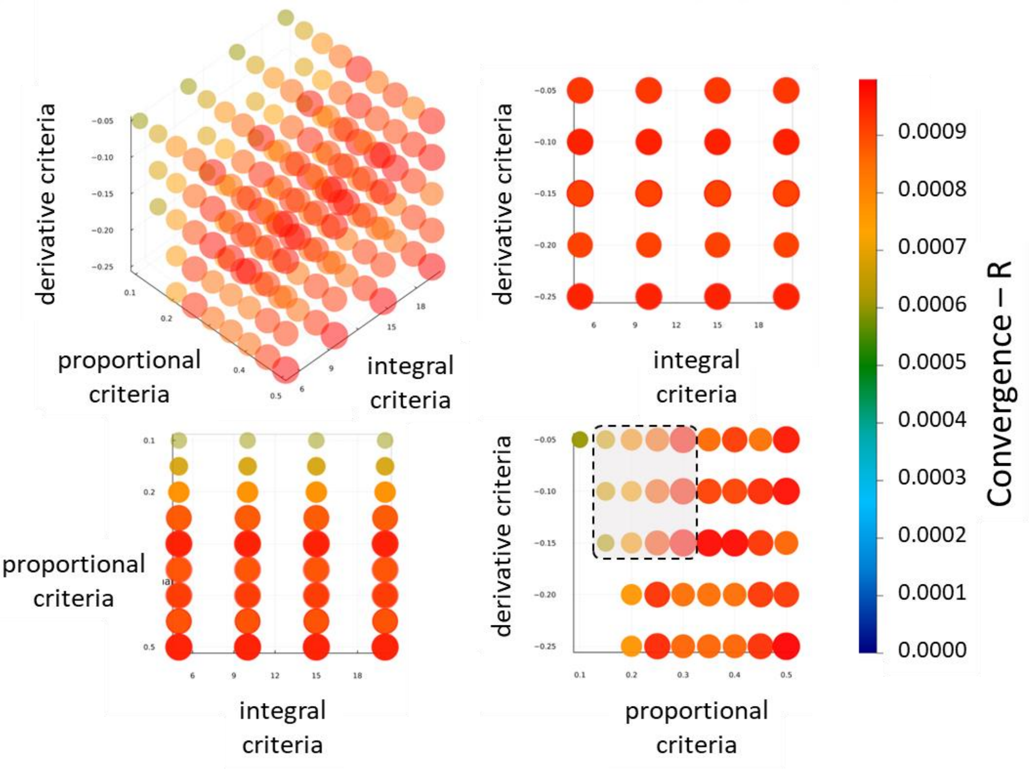


Figure 4.60. *Response points and orthogonal views of convergence of mean residual for exploration of PID driven cycle criteria at broad space*

High proportional criterion values ended up with relatively high cost as seen in Figure 4.59 while high values of derivative ended up with relatively low cost as seen in Figure 4.59. Integral criterion seemed no or little effect on cost as seen in both Figure 4.59 and Figure 4.60. Low proportional criterion values coupled with high derivative criterion values ended up with low convergence levels as seen in Figure 4.60 for mean residual.

According to results of initial exploration of PID driven cycle at broad space, following exploration at a narrower space was designed as seen in Figure 4.57 and Figure 4.58 as a region encompassed with dashed lines. Proportional criterion varied between 0.15 and 0.30 with 0.025 step; integral criterion varied between 4.0 and 7.0 with 1.0 step; derivative criterion varied between -0.15 and -0.05 with 0.02 step; for the exploration of the narrow space. Selected space for the following exploration had higher resolution than the initial exploration to uncover precise boundary of validity.

Metrics of the following exploration of PID driven cycle at narrow space that varied PID criteria were given in Table 4.23 and Table 4.24 for cost and convergence respectively. Response surface points and contour view of points of the exploration of PID driven cycle were given in Figure 4.61 and Figure 4.62 to visualise exploration metrics for cost and convergence respectively.

Table 4.23. *Exploration metrics of PID driven cycle with varying PID criteria regarding cost in reference work unit at narrow space (for integral criterion = 5.0)*

Cost (Integral Criterion = 5.0)		Derivative Criterion					
		-0.150	-0.130	-0.110	-0.090	-0.070	-0.050
Proportional Criterion	0.150	6.18	6.24	6.21	6.24	6.32	7.38
	0.175	6.08	6.11	6.26	6.19	6.26	7.35
	0.200	6.09	6.08	6.16	6.20	6.21	7.35
	0.225	6.16	6.06	6.14	6.14	6.18	11.07
	0.250	6.05	6.06	6.14	6.22	9.86	11.15
	0.275	6.17	6.06	6.13	6.15	9.85	11.03
	0.300	6.07	6.08	6.26	9.76	10.90	10.95

Table 4.24. *Exploration metrics of PID driven cycle with varying PID criteria regarding convergence of mean residual at narrow space (for integral criterion = 5.0)*

Convergence (Integral Criterion = 5.0)		Derivative Criterion					
		-0.150	-0.130	-0.110	-0.090	-0.070	-0.050
Proportional Criterion	0.150	0.00063	0.00063	0.00069	0.00068	0.00068	0.00068
	0.175	0.00069	0.00068	0.00069	0.00074	0.00074	0.00073
	0.200	0.00077	0.00080	0.00077	0.00074	0.00077	0.00079
	0.225	0.00084	0.00084	0.00088	0.00084	0.00083	0.00082
	0.250	0.00091	0.00090	0.00088	0.00086	0.00082	0.00087
	0.275	0.00098	0.00098	0.00096	0.00094	0.00089	0.00087
	0.300	0.00098	0.00098	0.00096	0.00091	0.00085	0.00097

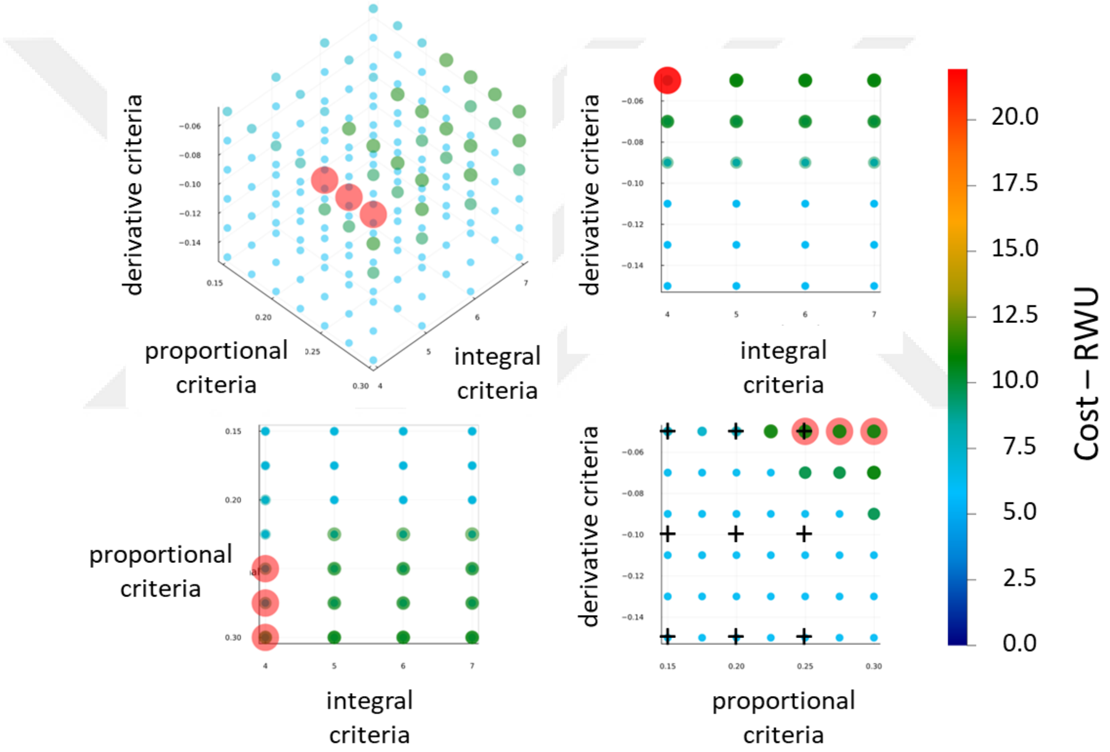


Figure 4.61. *Response points and orthogonal views of cost in reference work unit for exploration of PID driven cycle criteria at narrow space*

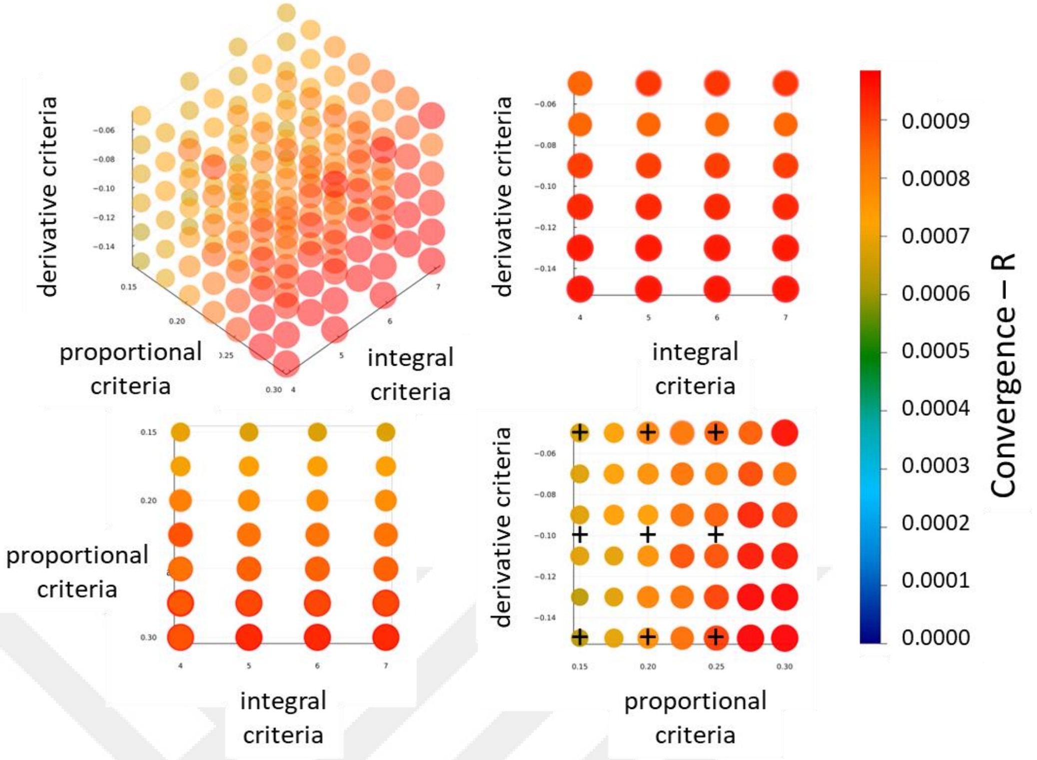


Figure 4.62. *Response points and orthogonal views of convergence of mean residual for exploration of PID driven cycle criteria at narrow space*

Cost of the PID driven multigrid cycle varied fairly regularly with changing PID criteria for exploration at narrow space as seen in Figure 4.61 and Figure 4.62. Convergence levels of mean residuals also varied regularly with changing PID criteria as seen in Figure 4.61 and Figure 4.62. Although proportional and derivative criterion values were effective on the outcome, integral criterion did not change cost nor convergence beyond a certain value. Condition check for integral criterion did indeed trigger within cycles but did not reflect any complications on outcome. Integral criterion acted as a fuse against over investment as originally intended. Several sets of PID criteria were selected for further inspection as best performing sample points of exploration space.

Detailed cost metrics of PID driven multigrid cycle at selected points were given in Table 4.25 for various parameters with average values. Reference work unit (RWU) variation is the percentage difference between RWU measurement of the instance and the mean of the measurements. Iteration cost is the percentage cost of the iterations within the cycle. Finite volume cost is the percentage cost of the finite volume method functions within the cycle. Intergrid cost is the percentage cost of the intergrid operations (restriction, prolongation) within the cycle. Total costs of the cycles were given in Figure 4.63 alongside the performances of the PID driven cycle at selected points.

Table 4.25. *Cost metrics of PID driven multigrid cycles at selected points of PID criteria*

RWU variation		Dc		
		-0.15	-0.10	-0.05
Pc	0.15	-0.14%	0.44%	-0.39%
	0.20	0.26%	1.42%	-0.59%
	0.25	0.25%	-0.63%	-0.62%

iteration cost		Dc		
		-0.15	-0.10	-0.05
Pc	0.15	86.9%	88.5%	90.2%
	0.20	88.3%	88.8%	90.4%
	0.25	88.5%	88.5%	90.0%

finite volume cost		Dc		
		-0.15	-0.10	-0.05
Pc	0.15	11.8%	10.1%	8.7%
	0.20	10.4%	10.0%	8.5%
	0.25	10.3%	10.3%	8.6%

intergrid cost		Dc		
		-0.15	-0.10	-0.05
Pc	0.15	1.3%	1.3%	1.1%
	0.20	1.3%	1.2%	1.1%
	0.25	1.3%	1.3%	1.4%

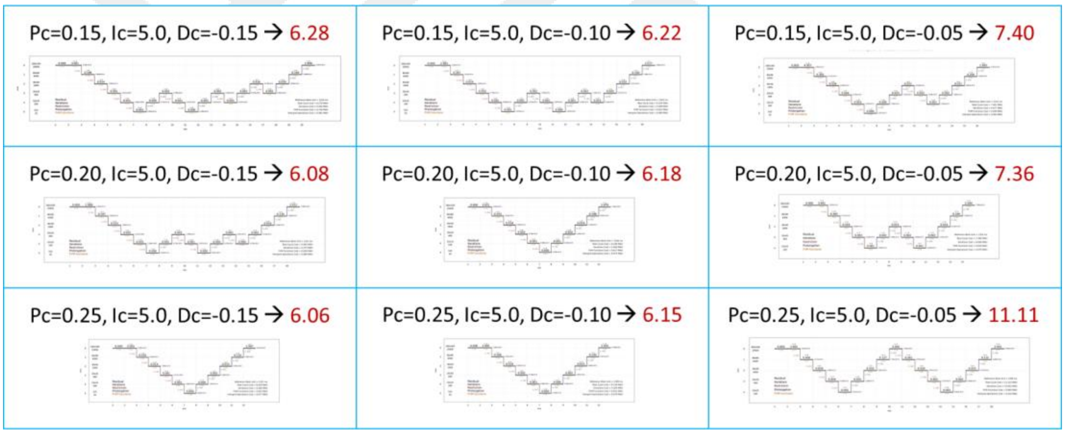


Figure 4.63. *Performances of PID driven multigrid cycles at selected points of PID criteria*

Best cases of PID driven cycle criteria exploration resided within the region that had proportional criterion values between 0.15 and 0.25; derivative criterion values between -0.15 and -0.05 for the reference case. Integral criterion was selected constant as 5.0 for the best cases of PID driven cycle criteria exploration for the reference case. Best performing PID driven cycles with selected criteria costed 6.985 reference work unit on average while having 0.000773 convergence level of mean residual. Run summary plots of the PID driven cycles with selected criteria were given in Appendix #7 for further inspection.

Exploration of PID criteria established the foundation of PID driven cycle for the reference case. PID criteria baseline was selected as 0.20, 5.0 and -0.10 for proportional, integral and derivative criterion respectively. Having the foundation and baseline for PID criteria enabled the exploration to tune PID coefficients for the reference case.

Proportional coefficient (cP) values were initially selected varying between 0.5 and 1.5 with 0.2 steps. Integral coefficient (Ic) values were initially selected varying between 0.5 and 1.5 with 0.2 steps. Derivative coefficient (Dc) values were initially selected varying between 0.5 and 1.5 with 0.2 steps. Initial grid resolution was selected as 160x160, so minimum grid resolution is 5x5 at coarsest level. Since the cycle scheme was adaptive, operational limitations were set. Minimum iterations per sweep was 2, maximum iterations per sweep was 50, maximum number of steps was 200 and final convergence target was 0.001 mean residual.

Metrics of the initial exploration of PID driven cycle at broad space that varied PID coefficients were given in Table 4.26 and Table 4.27 for cost and convergence respectively. Response surface points and contour view of points of the exploration of PID driven cycle were given in Figure 4.64 and Figure 4.65 to visualise exploration metrics for cost and convergence respectively.

PID driven cycles had tendency to stuck in a restriction and prolongation loop at various grid resolution levels for some sets of PID coefficient and criteria while solving the reference case. Solutions are only acceptable at the initial resolution level. Thus, invalid runs that were not complying with operational limitations were excluded and were marked. There were 204 valid runs in of broad space with 216 points.

Table 4.26. *Exploration metrics of PID driven cycle with varying PID coefficients regarding cost in reference work unit at broad space (for integral coefficient = 1.1)*

Cost (Integral Coef. = 1.1)		Derivative Coefficient					
		0.5	0.7	0.9	1.1	1.3	1.5
Proportional Coefficient	0.5	7.04	6.04	13.37	13.33	10.79	10.84
	0.7	7.05	6.04	6.10	9.71	10.81	10.86
	0.9	6.05	6.04	6.13	6.14	6.19	6.19
	1.1	6.13	6.07	6.22	6.20	6.26	6.26
	1.3	🟥	6.18	6.24	6.23	6.29	6.29
	1.5	🟥	6.15	6.23	6.21	6.29	6.29

Table 4.27. *Exploration metrics of PID driven cycle with varying PID coefficients regarding convergence of mean residual at broad space (for integral coefficient = 1.1)*

Convergence (Integral Coef. = 1.1)		Derivative Coefficient					
		0.5	0.7	0.9	1.1	1.3	1.5
Proportional Coefficient	0.5	0.00085	0.00098	0.00091	0.00090	0.00100	0.00097
	0.7	0.00085	0.00098	0.00096	0.00082	0.00086	0.00085
	0.9	0.00085	0.00084	0.00083	0.00084	0.00084	0.00083
	1.1	0.00072	0.00069	0.00069	0.00074	0.00074	0.00074
	1.3	🟥	0.00063	0.00068	0.00068	0.00068	0.00068
	1.5	🟥	0.00063	0.00067	0.00067	0.00063	0.00063

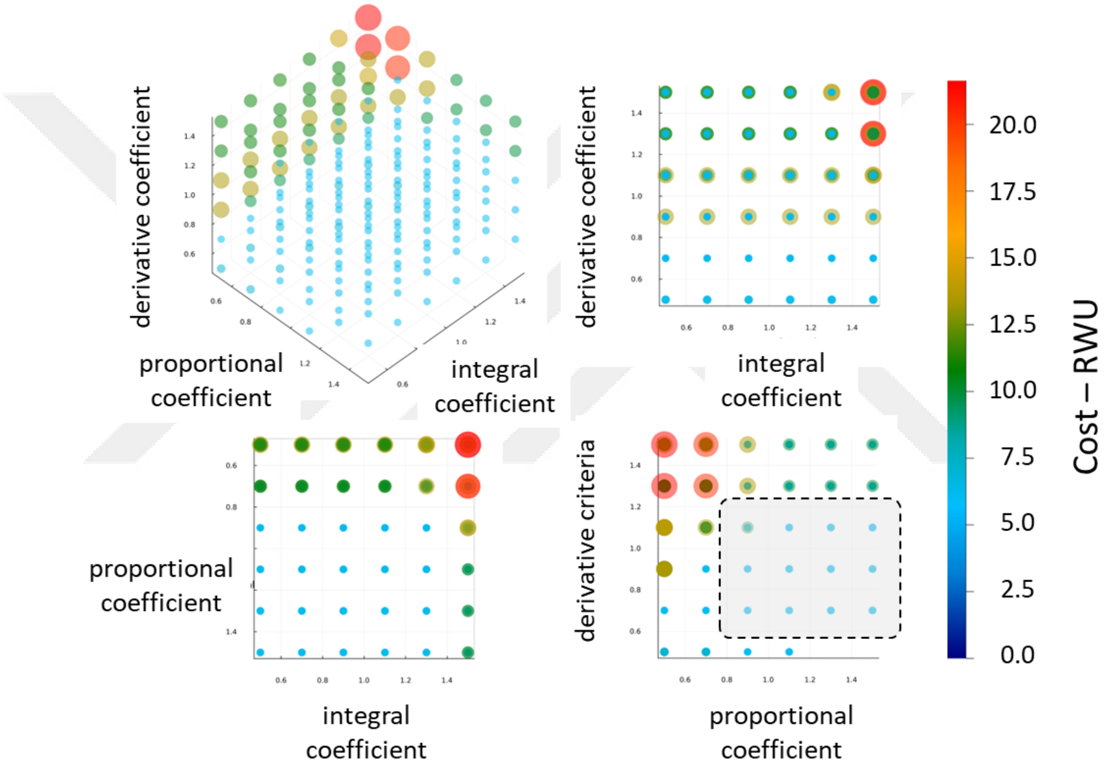


Figure 4.64. *Response points and orthogonal views of cost in reference work unit for exploration of PID driven cycle coefficients at broad space*

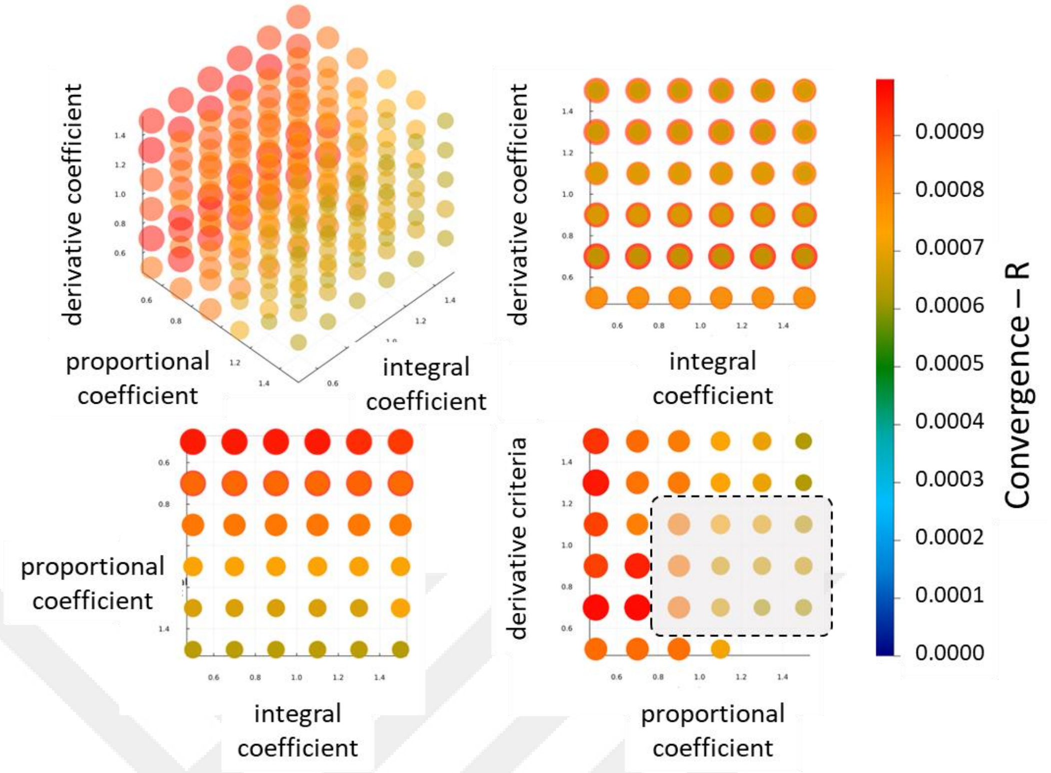


Figure 4.65. *Response points and orthogonal views of convergence of mean residual for exploration of PID driven cycle coefficients at broad space*

High proportional coefficient values ended up with relatively low cost as seen in Figure 4.64 while low values of derivative coefficient ended up with relatively low cost as seen in Figure 4.64. Integral criterion seemed no or little effect on cost as seen in both Figure 4.64 and Figure 4.65. Low proportional coefficient values coupled with high derivative coefficient values ended up with distinctive high cost as seen in Figure 4.64.

According to results of initial exploration of PID driven cycle at broad space, following exploration at a narrower space was designed as seen in Figure 4.64 and Figure 4.65 as a region encompassed with dashed lines. Proportional coefficient varied between 0.8 and 1.6 with 0.1 step; integral coefficient varied between 0.6 and 1.4 with 1.0 step; derivative coefficient varied between 0.6 and 1.2 with 0.1 step; for the exploration of the narrow space. Selected space for the following exploration had higher resolution than the initial exploration to uncover precise boundary of validity.

Metrics of the following exploration of PID driven cycle at narrow space that varied PID coefficients were given in Table 4.28 and Table 4.29 for cost and convergence respectively. Response surface points and contour view of points of the exploration of

PID driven cycle were given in Figure 4.66 and Figure 4.67 to visualise exploration metrics for cost and convergence respectively.

Table 4.28. *Exploration metrics of PID driven cycle with varying PID coefficients regarding cost in reference work unit at narrow space (for integral coefficient = 1.0)*

Cost (Integral Coef. = 1.0)		Derivative Coefficient						
		0.6	0.7	0.8	0.9	1	1.1	1.2
Proportional Coefficient	0.8	6.05	6.05	6.13	6.11	6.12	6.13	6.18
	0.9	6.05	6.06	6.12	6.12	6.11	6.11	6.17
	1	6.06	6.07	6.11	6.15	6.17	6.20	6.26
	1.1	6.13	6.09	6.21	6.22	6.19	6.18	6.26
	1.2	6.18	6.16	6.23	6.26	6.20	6.20	6.24
	1.3	6.22	6.17	6.22	6.25	6.21	6.23	6.28
	1.4	6.21	6.16	6.20	6.18	6.18	6.20	6.26
	1.5	6.20	6.16	6.20	6.23	6.23	6.22	6.29
	1.6	6.20	6.18	6.22	6.23	6.22	6.22	6.29

Table 4.29. *Exploration metrics of PID driven cycle with varying PID coefficients regarding convergence of mean residual at narrow space (for integral coefficient = 1.0)*

Convergence (Integral Coef. = 1.0)		Derivative Coefficient						
		0.6	0.7	0.8	0.9	1	1.1	1.2
Proportional Coefficient	0.8	0.00091	0.00091	0.00090	0.00088	0.00088	0.00086	0.00086
	0.9	0.00084	0.00084	0.00084	0.00083	0.00083	0.00084	0.00084
	1	0.00077	0.00077	0.00080	0.00077	0.00075	0.00074	0.00074
	1.1	0.00072	0.00069	0.00072	0.00069	0.00075	0.00074	0.00074
	1.2	0.00064	0.00064	0.00068	0.00068	0.00075	0.00074	0.00074
	1.3	0.00061	0.00063	0.00068	0.00068	0.00073	0.00068	0.00068
	1.4	0.00060	0.00063	0.00069	0.00069	0.00069	0.00069	0.00069
	1.5	0.00061	0.00063	0.00063	0.00067	0.00067	0.00067	0.00067
	1.6	0.00060	0.00063	0.00062	0.00063	0.00063	0.00063	0.00063

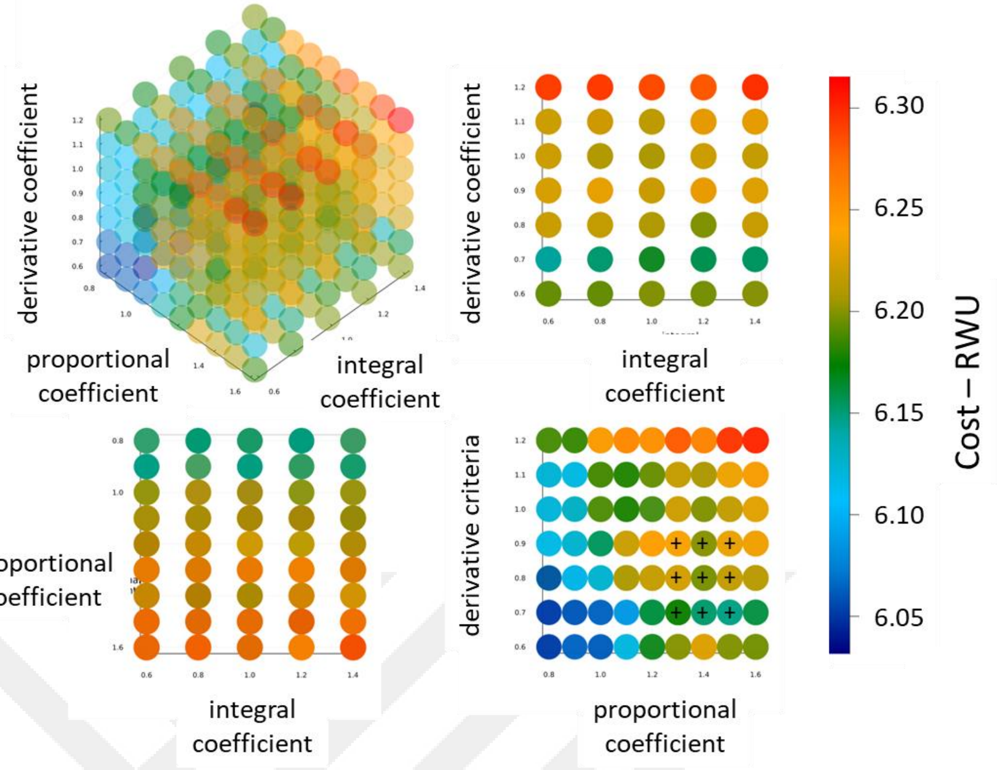


Figure 4.66. *Response points and orthogonal views of cost in reference work unit for exploration of PID driven cycle coefficients at narrow space*

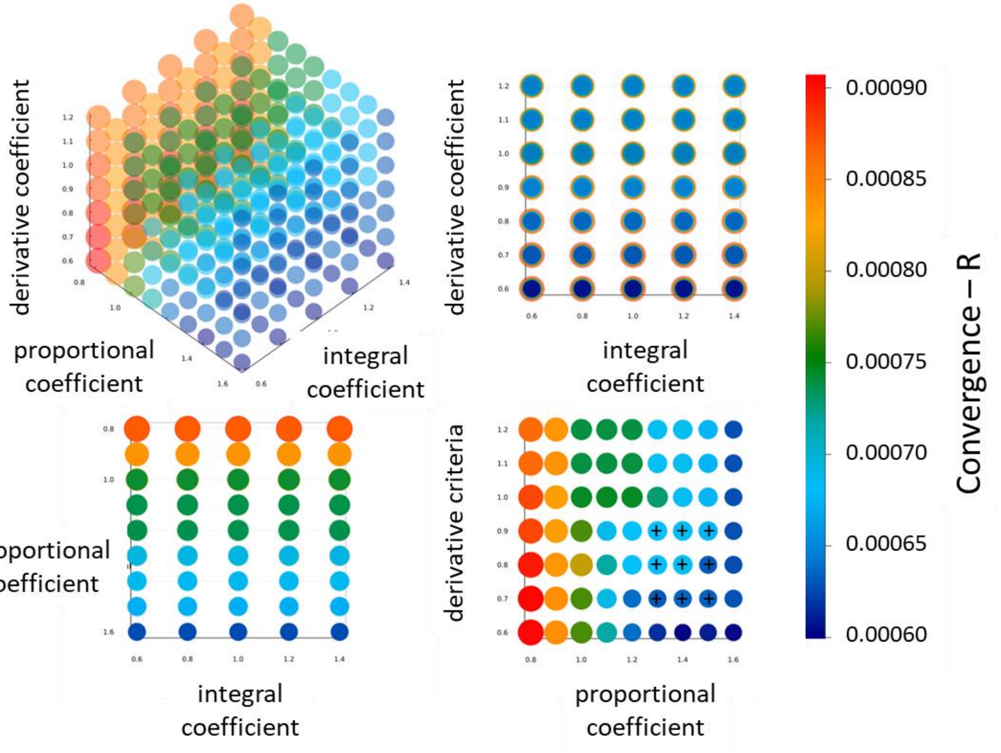


Figure 4.67. *Response points and orthogonal views of convergence of mean residual for exploration of PID driven cycle coefficients at narrow space*

Cost of the PID driven multigrid cycle varied fairly regularly but at a small amount with changing PID coefficient for exploration at narrow space as seen in Figure 4.66 and Figure 4.67. Convergence levels of mean residuals also varied regularly with changing PID coefficient as seen in Figure 4.66 and Figure 4.67. Although proportional and derivative coefficient values were effective on the outcome, integral coefficient did not change cost nor convergence beyond a certain value just like integral criterion. Several sets of PID criteria were selected for further inspection as best performing sample points of exploration space.

Detailed cost metrics of PID driven multigrid cycle at selected points were given in Table 4.25 for various parameters with average values. Reference work unit (RWU) variation is the percentage difference between RWU measurement of the instance and the mean of the measurements. Iteration cost is the percentage cost of the iterations within the cycle. Finite volume cost is the percentage cost of the finite volume method functions within the cycle. Intergrid cost is the percentage cost of the intergrid operations (restriction, prolongation) within the cycle. Total costs of the cycles were given in Figure 4.68 alongside the performances of the PID driven cycle at selected points.

Table 4.30. *Cost metrics of PID driven multigrid cycles at selected points of PID coefficients (for integral coefficient = 1.0)*

RWU variation		cD		
		0.7	0.8	0.9
cP	1.3	-1.3%	1.0%	1.0%
	1.4	-0.2%	0.0%	0.3%
	1.5	-0.2%	-0.2%	-0.5%

iteration cost		cD		
		0.7	0.8	0.9
cP	1.3	88.5%	88.7%	88.5%
	1.4	88.2%	88.8%	89.0%
	1.5	88.6%	88.5%	88.6%

finite volume cost		cD		
		0.7	0.8	0.9
cP	1.3	10.2%	10.1%	10.0%
	1.4	10.1%	10.0%	9.8%
	1.5	10.2%	10.3%	9.9%

intergrid cost		cD		
		0.7	0.8	0.9
cP	1.3	1.3%	1.2%	1.5%
	1.4	1.7%	1.2%	1.2%
	1.5	1.2%	1.2%	1.5%

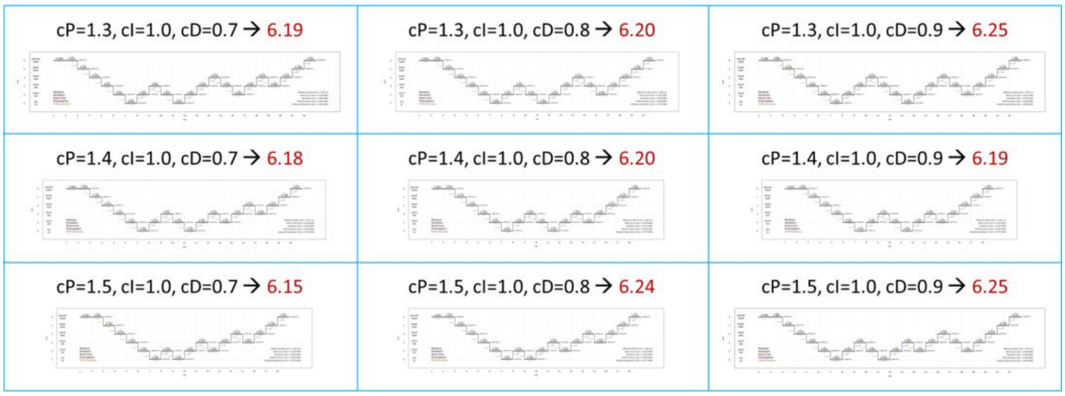


Figure 4.68. *Performances of PID driven multigrid cycles at selected points of PID coefficients (for integral coefficient = 1.0)*

Best cases of PID driven cycle coefficient exploration resided within the region that had proportional coefficient values between 1.3 and 1.5; derivative coefficient values between 0.7 and 0.9 for the reference case. Integral criterion was selected constant as 1.0 for the best cases of PID driven cycle coefficient exploration for the reference case. Best performing PID driven cycles with selected coefficients costed 6.208 reference work unit on average while having 0.000659 convergence level of mean residual. Run summary plots of the PID driven cycles with selected coefficients were given in Appendix #8 for further inspection.

4.4. Summary of Results

Thermal diffusion on a homogenous plate was studied and was assumed as reference case. Analytical Laplace solution of the reference case was present to compare with solutions of direct iterative method and multigrid cycles with fixed and adaptive schemes. Validation of direct iterative and multigrid cycle solutions were shown with statistical analyses. Solution progress was examined via residual progress and residual distribution within the domain of solutions. Residual progress of solutions on different grid resolutions were explored while inspecting progress of property distribution as well. Progresses of mean and quartile values of solution were also examined. Elapsed time values per iteration were measured for different grid resolutions. Reference work unit relation as a function of number of cells was founded for the reference case. Various direct iterative solutions were generated with differing Dirichlet and Neumann boundary conditions. Effect of initialization was explored for different grid resolutions. Cost and convergence levels of multigrid cycles with fixed schemes were explored with varying iterations per sweep and maximum coarse grid level. Reference multigrid cycle was defined according to the results of the exploration of cycles with fixed schemes. Cost and convergence levels of multigrid cycles with adaptive schemes were explored with varying input parameters. Flexible cycles were explored with varying alpha and beta parameters. PID driven cycles were explored with varying PID coefficients after the exploration of and establishing of PID criteria.

Analytical solution of the reference case enabled the calculation of the property at any location within domain regardless of the grid resolution. Statistical analyses showed that all methods are valid and generates correct solutions with appropriate parameters. Progress of mean residual and property distribution revealed that solution accuracy is only acceptable when a certain convergence for mean residual is reached. Exploring solution progress on different grid resolutions revealed that quartiles (distribution) converge differently for each grid resolution. Examining measurements of elapsed time values per iteration revealed that time dependant derivatives are not reliable for steering mechanisms. Exploring the effect of different boundary conditions revealed that main contributor to cost is high gradients rather than different boundary conditions. Additionally, the approach to initialization by averaging Dirichlet boundary conditions were found convenient when there are no Neumann boundary conditions present. Initialization with values closer to actual solution is observed to have a great effect on reducing cost. Exploration of multigrid cycles with fixed schemes revealed that there is a minimum number of iterations per sweep. Multigrid cycle cost increases with iterations per sweep while maximum coarse level does not affect cost. On the other hand, maximum coarse level is much more effective than iterations per sweep to obtain better convergence levels. Exploring adaptive cycles with respective input parameters revealed that adaptive cycles are not completely robust compared to cycles with fixed schemes. On the other hand, adaptive cycles assure and most of the time exceed desired convergence levels if they are not stuck in steering loops. Effectivity of both flexible and PID driven cycle are found almost identical while PID driven cycle is more robust on a broader range of input parameters.

Adaptive multigrid cycles were the main interest of the thesis while cost and convergence were main focal points. Performance of the adaptive multigrid cycles were measured with total cost in reference work units. High performing multigrid cycle was assumed to have low cost. Solution accuracy of the adaptive multigrid cycles was measured with convergence level in mean residual. Accurate solution was assumed to have low mean residual value. Effective multigrid cycle was assumed to get an accurate solution with low cost. In this manner, effectivity indicator was defined as multiplication of convergence and cost in order to combine both outcomes. Relatively low value of effectivity indicator corresponds to relatively more effective multigrid cycle. Effectivity indicator is for only comparison purposes. Average values regarding performance and accuracy were given in Table 4.31 to emphasize high level comparison of adaptive multigrid cycles with respect to reference multigrid cycle. Average values presented in Table 4.26 was derived using the data given in exploration results of selected best cases of respective cycles. Total cost ratios are percentage of total cost values to total cost value of reference multigrid cycle. Total cost values are the measured costs in reference work unit. Iteration cost ratio, finite volume method (FVM) cost ratio and intergrid operation cost ratio are respective percentages of cost within the total cost values. Convergence levels are the mean residuals of solutions. Effectivity indicator values are calculated as defined.

Table 4.31. *High level comparison of adaptive multigrid cycles with respect to reference multigrid cycle*

Average Values	Unit	Reference Multigrid Cycle	Flexible Cycle (at best)	PID Driven Cycle (at best)
Total Cost Ratio	%	100 %	152 %	124 %
Total Cost	Reference Work Unit	5.00	7.60	6.21
Iteration Cost Ratio	%	85.0 %	90.6 %	88.6 %
FVM Cost Ratio	%	13.0 %	8.4 %	10.1 %
Intergrid Cost Ratio	%	2.0 %	1.0 %	1.3 %
Convergence Level	Mean Residual	0.001373	0.000523	0.000659
Effectivity Indicator	Convergence * Cost	0.006865	0.00397	0.00409

Essential aspects of the results were reflected on Table 4.31 that presents clear inferences. PID driven cycle costed less compared to Flexible cycle; for a desired convergence target of 0.001 and same operational limitations. Both adaptive cycles

(Flexible and PID driven) assured and exceeded convergence target as seen in respective convergence levels. Both adaptive cycles had similar cost percentages of iterations, FVM and intergrid operations while PID driven cycle had tendency to have slightly more FVM and intergrid operations compared to Flexible cycle. Having slightly more FVM and intergrid operations means that PID driven cycle tends to steer more at coarse to mid grid resolution levels. Both adaptive cycles had almost identical effectivity indicator values. Both adaptive cycles were much more effective than the reference multigrid cycle.

5. CONCLUSION

Lack of variety of adaptive multigrid schemes were spotted and motivated the thesis. In order to justify a fair numerical experimentation setup, authentic code generation effort was given that implements basic finite volume methods, intergrid operations and iterative solvers. A fundamental reference case was selected in order to get a proper validation. Multigrid schemes and algorithms were also implemented. All methods were presented with both theory and practice in a harmonized manner. All procedures and algorithms were validated regarding the reference case. Results were compiled and presented in an increasingly complex order. Innovative aspect of the thesis was carried out by developing a new and an alternative adaptive multigrid scheme. Comparative performances of multigrid schemes revealed that newly developed adaptive multigrid scheme is robust and is performing as well as the other ones if not better.

Many aspects of multigrid methods and CFD code generation were revealed by observations throughout the journey. In order to present key points and discuss findings in an organized manner, experimental design approach was used [43]. Brief reminders to terminology were given in the following paragraphs accompanied by illustrations.

Figure 5.1 illustrates variables of a generic process. Input variables [43] are the factors or quantities that are given or selected at the beginning of a process. They are the initial conditions or parameters that influence the process but are not directly affected by it. Controllable variables [43] are the factors that the experimenter or system can adjust or control during the process. Changes in these variables are intentional and designed to observe their impact on the response variables. Uncontrollable variables [43] are factors that may influence the process but are beyond the control of the experimenter. They can introduce variability and uncertainties into the system, making it important to account for them when analysing the results. Response variables [43]; also known as output variables, are the measurable outcomes or results of a process. They represent the effects or changes caused by the processing of input variables.

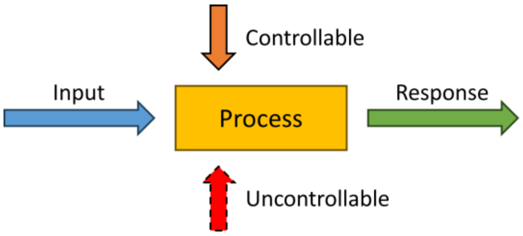


Figure 5.1. *Variable diagram of a generic process*

Figure 5.2 illustrates cause and effect diagram regarding exploration of adaptive multigrid cycles. Most of the factors were not clearly apparent when study started and most of them were also seemed as controllable if not constant at all. However, journey through thesis revealed that Figure 5.2 is indeed factual according to observations and results.

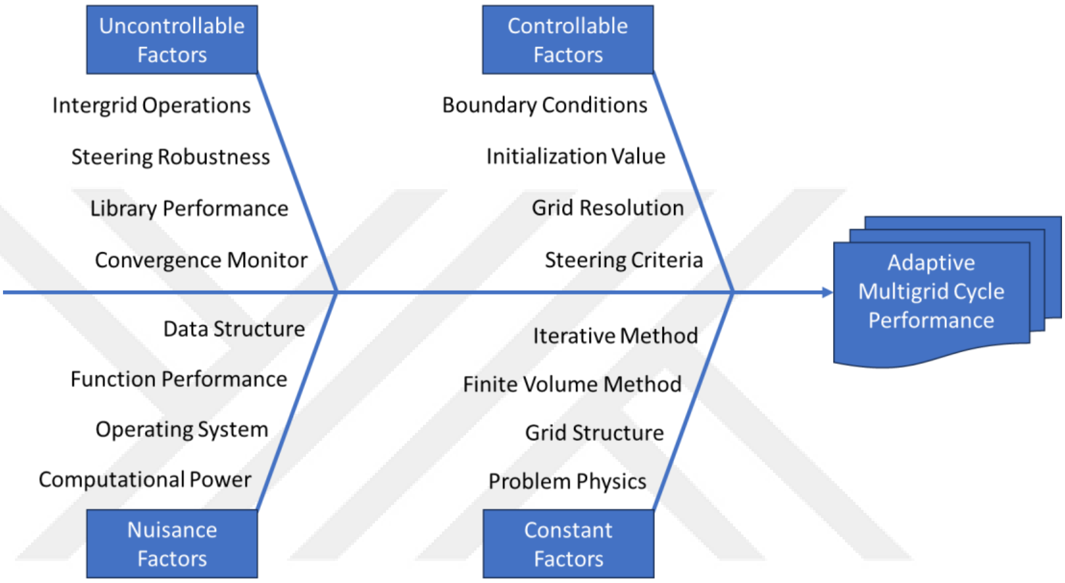


Figure 5.2. *Cause and effect diagram for exploration of performance of adaptive multigrid cycles*

Revelations regarding the concepts presented in Figure 5.2 are explained in observation and discussion sections. Future work section presents insights about migrating some factors from one type to another.

5.1. Observations

Many findings led to several insights regarding iterative solvers, finite volume methods, multigrid cycles as well as software development. Some of the noteworthy observations made during studies were given even if they are not directly related to main objective.

Time dependent derivatives are unusable with a lot of uncertainty and random spikes for any sort of condition checks or reasoning.

Computational cost dramatically changes with data structure. Memory and CPU usage spikes through the roof with dense matrix types. Sparse matrices are almost an obligation for any CFD code.

Procedures and algorithms implemented within functions (such as Gauss-Seidel iterator, intergrid operations, finite volume methods, etc...) directly effects computational cost as well as performance. Software engineering principles also apply to any CFD code imperatively.

System (& data type) epsilon and system response are physical limiting factors on performance. No algorithm nor procedure can perform beyond system capabilities. System consists of hardware and software. Software consists of operating system, language environment and code itself.

Reference work unit relation above certain cell number indicates that with increasing problem volume exponential computational cost growth is indeed natural and is unavoidable.

Most of the computational cost is expensed on the iterations and operations on initial/finest grid level. Any effort that reduces the cost of initialization and finalization would reduce overall cost dramatically.

Convergence criteria directly changes required level of investment for solution. Mean residual is one of the most smooth and practical convergence monitors. Statistical monitors like quartile residuals are unusable with highly oscillating behaviour in order to rely for an adaptive scheme.

Steering criteria is the very essence of any adaptive multigrid scheme. Inherently sequential and careful evaluation of criteria and condition checks are the very foundation of a robust adaptive multigrid scheme. Convergence is assured by adaptive multigrid schemes if they are not stuck in a steering induced (intergrid operation) loop.

5.2. Discussion

Discussion of the key findings were given on the basis illustrated in Figure 5.2. Inferences and revelations were shared qualitatively while quantitative outcomes were already given in results section.

Constant factors of the setup can be listed as; iterative method, finite volume method, grid structure and problem physics. These factors were selected as constant in order to reduce complexity of the setup and to focus the effort given on main objective.

Iterative method (unmodified Gauss Seidel without any relaxation) was held constant in order to avoid complexity input of advanced iterative methods. Selecting a basic iterative method enabled to get a proper response to steering criteria input. In this manner, iterations became a cost item rather than iterative method.

Finite volume method procedures were kept constant in order to eliminate cost contribution of varying discretisations. Selecting a FVM approach enabled to get a proper response to boundary condition and grid resolution input. By doing so, cost contributions of FVM procedures were separated and steering criteria was promoted instead as a cost item.

Grid type or structure was kept constant in order to filter out grid variations which have same number of cells. With same type and structure, grids with same number of cells should be identical for the domains with same sizes. Thus, grid contribution was narrowed down to number of cells input. By fixing the structure, grid resolution became a distinguishable variable.

Problem physics (reference case) was kept constant in order to focus efforts on multigrid cycles while eliminating the variations of procedures. With changing problem physics and/or phenomena finite volume methods may variate too. Instead of adding another layer or dimension to the setup, problem physics was limited to a single case type. Since problem physics changes the very foundation of methods towards solution, every problem type should be explored in a separate experimental setup.

Nuisance factors of the setup can be listed as; data structure, function performance, operating system and computational power. Some of the nuisance factors were also mentioned in observation section. Even though most of the nuisance factors were blocked by the setup, slight deviations on cost and performance were also detected.

Data structure was fixed for all runs. However, in conjunction with other nuisance factors, even after some code optimization effort, without advanced software engineering, data structure became one of the limiting factors preventing solution of cases with high number of cells. Solutions of systems of equations are iterated over coefficient matrices and vectors. Even with sparse matrices approach, without unlimited resource, case volume was limited. Thus, data structure factor was blocked in order to demote it to be a mere constraint.

Function performance was another nuisance factor that was not completely blocked. Even though procedures were exactly same, varying conditions were detected to induce slight differences in terms of performance. Intergrid operations on same grid resolution were observed to have slightly different performances depending on convergence level of the solution delivered to function. Finite volume methods also displayed same behaviour with differing boundary conditions. Iterative methods seemed fairly consistent with changing input. Since both FVM and intergrid operations consisted low percentage of the total cost, function performance contributions were considered as nuisance factors.

Operating system was another source of uncertainty with background processes causing random and variant loads causing spikes on latency and/or response of functions. Random behaviour of these spikes induced great effect on processes with short term runtime which disrupted accurate measurements. Processes with long term runtime were also affected with relatively negligible amount but may be affected by more than a singular spike. In any case, uncertainty contribution of operating system was blocked by either increasing sampling or using measurements with spikes having negligible contribution percentage.

Computational power may be seemed as a constant factor but in fact a nuisance factor. Combined with operating system and the environment in which the code runs, computational power fluctuated with varying resource allocation of the machine. Additionally, resource allocation also varied with hardware and software configuration. Even though this effect was limited within fixed configuration; it was blocked by using a non-dimensional (reference work unit) performance parameter.

Uncontrollable factors of the setup can be listed as, intergrid operations, steering robustness, library performance and convergence monitor. Effect of these factors were detected on cost and performance and could not be mitigated.

Intergrid operations induced a jump on mean residual of intermediate solution which seemed to be varying with conditions. This setback hindered convergence rate at an unknown amount. Since the relation between conditions and setback was not revealed in this study, setback effect of intergrid operations was accepted as an uncontrollable factor for multigrid cycle performance. Furthermore, contribution of intergrid operation cost with respect to total cost was very low. Thus, steering criteria factor represented this contribution instead.

Steering robustness could not be assured for any parameter and/or factor set of an adaptive multigrid cycle. Some of the parameters induced steering loops for some cases. Properties and/or conditions that induced steering loop in conjunction with steering criteria was elusive to specify. Some parameters were simply not compatible with some other factors (such as boundary conditions, grid resolution and steering criteria) to properly steer through. It almost feels like algorithm lost grip and skid when some parameters met some conditions. It was possible to mitigate steering robustness issue with hard counter measures but it would come with the cost of losing the very essence of adaptive multigrid schemes. Thus, steering robustness issue was accepted as an uncontrollable factor for adaptive multigrid cycles. Runs that had steering induced loops were marked and left out of evaluation.

Library performance issue was arisen from the implementation of methods within the libraries that were used. Both external and internal libraries employs and/or offers limited variety of procedures for the same task. Alternating between the approaches to carry out same task differs in terms of performance. Library performance also varied with condition. Since main objective was not code optimization, best available usages of libraries were assumed and were accepted as uncontrollable factor.

Convergence monitor concept is mean residual. Monitoring an alternative concept for an adaptive multigrid scheme was tempting. However, both statistical and derivative evaluation of residual vector and/or progress proved unusable according to observations. Using mean residual as convergence monitor also promoted consistency of newly developed adaptive scheme. In this manner, convergence monitor was accepted as uncontrollable factor even though it may be changed with various complications.

Controllable factors can be listed as; boundary conditions, initialization value, grid resolution and steering criteria. These factors could be varied and in fact were varied within explorations. Exploration chapter of results section presents responses of these factors thoroughly. Simply put, with all the assumptions, observations, efforts and findings led to the experimental setup with these factors being controllable. In fact, thesis was built upon the most important controllable factor which is steering criteria. Discussion of these factors concludes thesis.

Grid resolution, boundary conditions and initialization value factors were explored through direct iterative solutions. Exploring these factors enabled narrowing factors further down in order to properly evaluate steering criteria of adaptive cycles. In the end, all previous effort to control factors other than steering criteria factor was the basis to suggest a new adaptive scheme and to compare it with the existing one.

Mean residual progress and property quartile progress on different grid resolutions were inspected. Response metrics of the direct iterative solutions revealed various relations. With increasing grid resolution, increasing cost to get a solution was observed. Similarly, reference work unit (elapsed time per iteration) also increased with increasing grid resolution. Property quartile progress varied with varying grid resolution. Quartile of solution vectors converged faster on relatively coarse grids. Furthermore, a quadratic polynomial relation between number of cells and reference work unit could be written for grid resolutions above a certain level for a system (hardware and software) configuration.

Varying Dirichlet and Neumann boundary conditions revealed different implications on solution progress. Changing values of boundary conditions did not directly change cost. In fact, it was observed that the increasing percentage and/or number of high gradient regions within domain increased cost. Boundary conditions may indirectly increase cost by inciting high gradient regions. Boundary condition factor was explored with changing grid resolutions and it was observed that grid resolution is most more dominant in terms of cost with respect to boundary condition effect. The approach of initialization (by averaging Dirichlet boundary conditions) seemed to be effective at reducing cost when there was no Neumann boundary condition that incites high gradients on domain. Eventually, initialization value effect was to be explored.

Relatively correct initial guess of the solution appeared be effective at reducing cost as expected. Effect of cost reduction increased with increasing grid resolution. It was also observed that initial guess determines the approach direction of convergence towards solution.

Exploration of parameters of multigrid cycles with fixed schemes revealed additional insights towards establishing a knowledge basis for evaluating and/or developing an adaptive multigrid scheme. In order to get a valid solution from a multigrid cycle, there should be a minimum number of iterations per sweep at any resolution level. In order to reduce cost effectively, a multigrid cycle should visit coarse grid levels as much as possible. By considering these findings, a reference multigrid cycle that accumulates minimum cost was defined as a comparative reference for the adaptive multigrid cycles.

Steering criteria was the core of all experimental setup. In adaptive multigrid schemes steering criteria was the main distinguishing feature for both conceptually and practically. Thus, relation between the controllable factor (steering criteria) and response variables (cost and performance) was the ultimate revelation of the study. Clear presentation of this relation was built upon validated methods certify that in fact a new adaptive multigrid scheme was developed and was working properly which in return justified innovative aspect of the thesis.

Even though a steering algorithm was pretty straightforward to understand or to verbally describe, establishing a valid and robust one was simply proven challenging. A working algorithm needs quite a few robustness precautions embedded in the code. An adaptive multigrid cycle did not work with plain steering algorithm. Steering algorithm should be covered under a higher frame that deals with a lot of exceptions and conditions. Additionally, operational limitations were also should be in place in order to prevent algorithm to be stuck. Steering is a persistent state and/or possibility that can be triggered with each iteration. Thus, both adaptive multigrid cycle algorithm and iterative solution function that is called within needs to be in recursive structure. Shortly, it takes more than following a schematic illustration or pseudo-code to implement a working adaptive multigrid scheme, let alone develop a new one.

Since all other factors were either blocked or held constant, evaluation of responses and comparison of performances of adaptive multigrid cycles had valid basis with changing steering criteria. Thus, comparative statements can be given according to results between newly developed cycle (PID driven) and existing one (flexible). PID driven cycle costed less compared to flexible cycle; for a convenient convergence level and same operational limitations. Both adaptive cycles assured convergence target. Both adaptive cycles had almost identical effectivity indicators. Both adaptive cycles were found to be much more effective compared to the reference multigrid cycle. Newly developed PID driven cycle had broader feasible range regarding parameter inputs compared to existing flexible cycle, in terms of valid response (successful runs which delivers converged and correct solution).

Finally, according to results of the comprehensive numerical experimentation, it can be said that PID driven cycle is a newly developed and an alternative adaptive multigrid scheme, which works more robustly and performs evenly compared to existing adaptive multigrid schemes.

5.3. Future Work

Computational fluid dynamics is simply a vast field to cover. Multigrid methods alone have numerous studies and extensive literature. Total closure is practically impossible for a study regarding combination of CFD and multigrid. Efforts were focused to cover the main objective and a knowledge basis is established towards future work.

Convection with a pressure-velocity coupling algorithm may be the first improvement to the code suite. Domains with varying dimensions (1D, 2D, 3D) would be a great addition to the tool. FVM methods that could operate on non-uniform grids will enhance applicability to complex geometries. Advanced iterative methods such as Tridiagonal Matrix Algorithm (TDMA), Successive Over Relaxation (SOR) and Conjugate Gradient (CG) may improve performance by reducing the costs of initialization and finalization. Irregular or innovative multigrid schemes may be an interesting subject to study. Effect of basic operations (such as finite volume methods, intergrid operations, iterative methods) on cost and performance is an intriguing subject in terms of software engineering and may be explored thoroughly. Convergence criteria variations other than monitoring mean residual would be a challenging study. Implementation of parallelization to the code will enable the solution of problems with much higher grid resolutions. Code optimization in order to reduce computational cost will always be a never-ending task. Minimizing library dependency is always fruitful in terms of experience and performance. Less is more when it comes to dependencies. Robustness analyses of the adaptive multigrid schemes may be a tempting endeavour. Adaptive multigrid schemes may be tuned via neural network and/or machine learning procedures to utmost potency.

REFERENCES

[1] Versteeg H., Malalasekera W., 2007, *An Introduction to Computational Fluid Dynamics: The Finite Volume Method*, Pearson

[2] Sadrehaghighi I., 2023, *Essentials of CFD*, CFD Open Series

[3] Axelsson O., 1996, *Iterative Solution Methods*, Cambridge University Press

[4] Ferziger J.H., Peric M., 2002, *Computational Methods for Fluid Dynamics*, Springer

[5] Berger M.J., Oliger J., 1984, Adaptive mesh refinement for hyperbolic partial differential equations, *Journal of Computational Physics*, 53 (3), 484-512

[6] Tveito A., Bruaset A.M., (eds.), 2006, Numerical Solution of Partial Differential Equations on Parallel Computers, *Lecture Notes in Computational Science and Engineering*, 51, Springer

[7] Trottenberg U., Oosterlee C.W., Schüller A., 2001, *Multigrid*, Academic Press

[8] Brandt A., Livne O.E., 2011, *Multigrid Techniques: 1984 guide with applications to fluid dynamics*, Society for Industrial and Applied Mathematics

[9] McCormick S.F., (ed.), 1988, Multigrid Methods: Theory, Applications, and Supercomputing, *Lecture Notes in Pure and Applied Mathematics*, 110

[10] Wesseling P., 1992, *An Introduction to Multigrid Methods*, John Wiley & Sons

[11] Haase G., Langer U., 2002, Multigrid methods: from geometrical to algebraic versions, *Modern Methods in Scientific Computing and Applications*, 75. Springer

[12] Ciaramella G., Vanzan T., 2022, Substructured two-grid and multi-grid domain decomposition methods, *Numerical Algorithms*, 91, 413–448

[13] Bramble J.H., Zhang X., 2000, The Analysis of Multigrid Methods, *Handbook of Numerical Analysis*, 7, 173-415, Elsevier

[14] Brandt A., 1973, Multi-level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary Value Problems, *Lecture Notes on Physics*, 18, Springer

[15] Brandt A., 1977, Multi-Level Adaptive Solutions to Boundary-Value Problems, *Mathematics of Computation*, 31 (138), 333-390

[16] ANSYS Inc., 2020, *ANSYS Fluent Theory Guide*

[17] Siemens Digital Industries Software, 2023, *Simcenter STAR-CCM+ 2302 User Guide*

[18] Briggs W.L., Henson V.E., McCormick S.F., 2000, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics

[19] Blazek J., 2001, *Computational Fluid Dynamics: Principles and Applications*, Elsevier

[20] Wesseling P., 2009, *Principles of Computational Fluid Dynamics*, Springer

[21] Stüben K., Trottenberg U., 1982, Multigrid methods: Fundamental algorithms, model problem analysis and applications, *Lecture Notes in Mathematics*, 960, Springer

[22] Xu J., Zikatanov L., 2017, Algebraic Multigrid Methods, *Acta Numerica*, 2017, 591-721

[23] Martynenko S.I., 2017, *The Robust Multigrid Technique for Black-Box Software*, DeGruyter

[24] Jones J.E., McCormick S.F., 1997, Parallel Multigrid Methods, Parallel Numerical Algorithms, *Interdisciplinary Series in Science and Enginering*, 4, 203-224

[25] Chow E., Falgout R.D., Hu J.J., Tuminaro R.S., Yang U.M., 2006, A Survey of Parallelization Techniques for Multigrid Solvers, *Parallel Processing for Scientific Computing*, Society for Industrial and Applied Mathematics, 179-201

[26] McBryan O.A., Frederickson P.O., Lindenand J., Schüller A., Solchenbach K., Stüben K., Thole C.A., Trottenberg U., 1991, Multigrid methods on parallel computers A survey of recent developments, *IMPACT of Computing in Science and Engineering*, 3 (1), 1-75

[27] Falgout R.D., Jones J.E., 2000, Multigrid on Massively Parallel Architectures, Multigrid Methods VI., *Lecture Notes in Computational Science and Engineering*, 14, 101-107

[28] Hülsemann F., Kowarschik M., Mohr M., Rüde U., 2006, Parallel Geometric Multigrid, *Lecture Notes in Computational Science and Engineering*, 51, Springer

[29] AlMahdawi H.K., Sidikova A.I., Alkattan H., Abotaleb M., Kadi A., 2022, Parallel Multigrid Method for Solving Inverse Problems, *MethodX*, 9 (101887), Elsevier

[30] Mitchell W.F., 2004, Parallel Adaptive Multilevel Methods with Full Domain Partitions, *Applied Numerical Analysis & Computational Mathematics*, 1 (1), 36-48

[31] Yang U.M., 2006 Parallel Algebraic Multigrid Methods – High Performance Preconditioners, *Lecture Notes in Computational Science and Engineering*, 51, 209-236

[32] Brandt A., 2000 General Highly Accurate Algebraic Coarsening, *Electronic Transactions on Numerical Analysis*, 10

[33] Wesseling P., Oosterlee C.W., 1999, Geometric Multigrid with Applications to Computational Fluid Dynamics, *Journal of Computational and Applied Mathematics*, 128, 311-334

[34] Rhee J.R., Oh J.S., Huh J.Y., Kim K.H., 2023, Meshless Geometric Multigrid Method for Complex Geometries with Improved Cell Coarsening Algorithm, *AIAA Journal*, 0, 1-14

[35] Clevenger T.C., Heister T., Kanschat G., Kronbichler M., 2020, A Flexible, Parallel, Adaptive Geometric Multigrid Method for FEM, *ACM Transactions on Mathematical Software*, 47 (1), 1-27

[36] Saberi S., Meschke G., Vogel A., 2023, Adaptive geometric multigrid for the mixed finite cell formulation of Stokes and Navier–Stokes equations, *International Journal of Numerical Methods in Fluids*, 95 (7), 1035–1053

[37] Brown J., He Y., MacLachlan S., Menickelly M., Wild S.M., 2021, Tuning Multigrid Methods with Robust Optimization and Local Fourier Analysis, *SIAM Journal on Scientific Computing*, 43 (1), A109-A138

[38] Grasedyck L., Klever M., Löbbert C., Werhmann T.A., 2020, A Parameter-dependent Smoother for the Multigrid Method, *arXiv:2008.00927*

[39] Arge E., Bruaset A.M., Langtangen H.P., (eds.), 1997, *Modern Software Tools for Scientific Computing*, Springer

[40] Ogata K., 2010, *Modern Control Engineering*, Printice Hall

[41] Stewart J., Kokoska S., 2011, *Calculus: Concepts and contexts*, Cengage Learning

[42] Bureau International des Poids et Mesures, 2019, *The International System of Units (SI)*, 9th edition

[43] Montgomery D.C., 2017, *Design and Analysis of Experiments*, Wiley

[44] Patil P.V., Prasad K., 2014, Numerical Solution for Two Dimensional Laplace Equation with Dirichlet Boundary Conditions, *IOSR Journal of Mathematics*, 6, 66-75

http-1: https://julialang.org/benchmarks/ (accessed at 28.11.2023)

http-2: https://docs.julialang.org/ (accessed at 28.11.2023)

http-3: https://docs.julialang.org/en/v1/stdlib/Dates/ (accessed at 28.11.2023)

http-4: https://docs.julialang.org/en/v1/stdlib/Printf/ (accessed at 28.11.2023)

http-5: https://docs.julialang.org/en/v1/stdlib/SparseArrays/ (accessed at 28.11.2023)

http-6: https://docs.julialang.org/en/v1/stdlib/Statistics/ (accessed at 28.11.2023)

http-7: https://csv.juliadata.org/ (accessed at 28.11.2023)

http-8: https://dataframes.juliadata.org/ (accessed at 28.11.2023)

http-9: https://docs.juliaplots.org/ (accessed at 28.11.2023)

http-10: https://github.com/JuliaPlots/StatsPlots.jl (accessed at 28.11.2023)

APPENDIX

Appendix 1.1

Finite Volume Method Function → “initialize_domain()”

```
# initialization of a dummy domain with input dimensions and division

function
initialize_domain(dim1::Float64,dim2::Float64,out_n_div_x::Int64,out_n_div_y::Int64)

    out_n_cell = Int64( out_n_div_x * out_n_div_y )
    out_c_dx = dim1 / out_n_div_x
    out_c_dy = dim2 / out_n_div_y

    out_Cs=DataFrame(ID=collect(Int64,1:1:out_n_cell))
    out_Cs.i=repeat(collect(Int64,1:1:out_n_div_x),out_n_div_y)
    out_Cs.j=vec(rotl90(repeat(collect(Int64,1:1:out_n_div_y),1,out_n_div_x)))

    out_Cs.x = ( out_Cs.i .* out_c_dx ) .- (out_c_dx/2)
    out_Cs.y = ( out_Cs.j .* out_c_dy ) .- (out_c_dy/2)

    out_Cs.T = spzeros(Float64,out_n_cell)

    return out_Cs

end
```

Appendix 1.2

Finite Volume Method Function → “boundary_conditions()”

```
# defining boundary conditions

function
boundary_conditions(WB_fix::Float64,WB_flux::Float64,EB_fix::Float64,EB_flux::Float64
,
SB_fix::Float64,SB_flux::Float64,NB_fix::Float64,NB_flux::Float64)

    out_df = DataFrame()
    out_df.boundary = [ "W" , "E" , "S" , "N" ]
    out_df.fix = [ WB_fix , EB_fix , SB_fix , NB_fix ]
    out_df.flux = [ WB_flux , EB_flux , SB_flux , NB_flux ]

    return out_df

end
```

Appendix 1.3

Finite Volume Method Function → “apply_boundary_conditions()”

```
# applying predefined BCs to a dummy domain

function apply_boundary_conditions(in_df::DataFrame,bc_df::DataFrame)

    # in_n_cell = in_df.ID[end]
    in_n_div_x = in_df.i[end]
    in_n_div_y = in_df.j[end]

    out_n_div_x = in_n_div_x + 2
    out_n_div_y = in_n_div_y + 2
    out_n_cell = out_n_div_x * out_n_div_y

    out_c_dx = (in_df.x[end]-in_df.x[1])/(in_n_div_x-1)
    out_c_dy = (in_df.y[end]-in_df.y[1])/(in_n_div_y-1)

    out_df=DataFrame(ID=collect(Int64,1:1:out_n_cell))
    out_df.i=repeat(collect(Int64,1:1:out_n_div_x),out_n_div_y)
    out_df.j=vec(rotl90(repeat(collect(Int64,1:1:out_n_div_y),1,out_n_div_x)))

    out_df.x = ( out_df.i .* out_c_dx ) .- (out_c_dx/2) .- out_c_dx
    out_df.y = ( out_df.j .* out_c_dy ) .- (out_c_dy/2) .- out_c_dy

    out_df.T = spzeros(Float64,out_n_cell)
    out_df.q = spzeros(Float64,out_n_cell)

    k = 1.0 * 1000.0 # W/mK
    A = out_c_dx * 0.01 # c_dx=c_dy and thickness = 1cm
    dζ = out_c_dx # c_dx=c_dy
    a = k*A/dζ # constant coefficient within domain

    # west boundary
    out_df[!,:T] .= ifelse.(out_df[!,:i].==1, bc_df.fix[1], out_df[!,:T])
    out_df[!,:q] .= ifelse.(out_df[!,:i].==1, bc_df.flux[1], out_df[!,:q])

    # east boundary
    out_df[!,:T] .=   ifelse.(out_df[!,:i].==out_n_div_x, bc_df.fix[2], out_df[!,:T])
    out_df[!,:q] .=   ifelse.(out_df[!,:i].==out_n_div_x, bc_df.flux[2],
out_df[!,:q])

    # south boundary
    out_df[!,:T] .=   ifelse.(out_df[!,:j].==1, bc_df.fix[3], out_df[!,:T])
    out_df[!,:q] .=   ifelse.(out_df[!,:j].==1, bc_df.flux[3], out_df[!,:q])

    # north boundary
    out_df[!,:T] .= ifelse.(out_df[!,:j].==out_n_div_y,    bc_df.fix[4], out_df[!,:T])
    out_df[!,:q] .= ifelse.(out_df[!,:j].==out_n_div_y,    bc_df.flux[4],
out_df[!,:q])

    # source calculations
    out_df.Sp=spzeros(Float64,out_n_cell)
    out_df[!,:Sp] .= ifelse.(out_df[!,:T].==0.0, out_df[!,:Sp],
ones(Float64,out_n_cell).*(-2*a))
    out_df.Su=out_df.q .- out_df.Sp .* out_df.T

    return out_df

end
```

Appendix 1.4

Finite Volume Method Function → “discretised_form()”

```
# transforming a domain with BC into discretised form
# calculation of coefficients (aW, aE, aS, aN, Sp, aP, Su)

function discretised_form(in_df::DataFrame)

    in_n_cell = in_df.ID[end]
    in_n_div_x = in_df.i[end]
    in_n_div_y = in_df.j[end]

    out_n_div_x = in_n_div_x - 2
    out_n_div_y = in_n_div_y - 2
    out_n_cell = out_n_div_x * out_n_div_y

    out_c_dx = (in_df.x[end]-in_df.x[1])/(in_n_div_x-1)
    out_c_dy = (in_df.y[end]-in_df.y[1])/(in_n_div_y-1)

    k = 1.0 * 1000.0 # W/mK
    A = out_c_dx * 0.01 # c_dx=c_dy and thickness = 1cm
    dζ = out_c_dx # c_dx=c_dy
    a = k*A/dζ # constant coefficient within domain

    out_df=DataFrame(ID=collect(Int64,1:1:out_n_cell))
    out_df.i=repeat(collect(Int64,1:1:out_n_div_x),out_n_div_y)
    out_df.j=vec(rotl90(repeat(collect(Int64,1:1:out_n_div_y),1,out_n_div_x)))

    out_df.x = ( out_df.i .* out_c_dx ) .- (out_c_dx/2)
    out_df.y = ( out_df.j .* out_c_dy ) .- (out_c_dy/2)

    # assigning dummy columns
    out_df.aW=ones(Float64,out_n_cell) .* a
    out_df.aE=ones(Float64,out_n_cell) .* a
    out_df.aS=ones(Float64,out_n_cell) .* a
    out_df.aN=ones(Float64,out_n_cell) .* a
    out_df.Sp=spzeros(Float64,out_n_cell)
    out_df.aP=spzeros(Float64,out_n_cell)
    out_df.Su=spzeros(Float64,out_n_cell)

    # assigning west boundary coefficients (link cutting)
    out_df[!,:aW] .= ifelse.(out_df[!,:i].==1, 0.0, out_df[!,:aW])

    # assigning west boundary coefficients (link cutting)
    out_df[!,:aE] .= ifelse.(out_df[!,:i].==out_n_div_x, 0.0, out_df[!,:aE])

    # assigning west boundary coefficients (link cutting)
    out_df[!,:aS] .= ifelse.(out_df[!,:j].==1, 0.0, out_df[!,:aS])

    # assigning west boundary coefficients (link cutting)
    out_df[!,:aN] .= ifelse.(out_df[!,:j].==out_n_div_y, 0.0, out_df[!,:aN])

    # taking neighbouring subsets of in_df
    in_df_W = in_df[(in_df.i.<in_n_div_x-1) .& (in_df.j.<in_n_div_y) .& (in_df.j.>1),
:]
    in_df_E = in_df[(in_df.i.>2) .& (in_df.j.<in_n_div_y) .& (in_df.j.>1), :]
    in_df_S = in_df[(in_df.j.<in_n_div_y-1) .& (in_df.i.<in_n_div_x) .& (in_df.i.>1),
:]
    in_df_N = in_df[(in_df.j.>2) .& (in_df.i.<in_n_div_x) .& (in_df.i.>1), :]

    # calculation of Sp coefficients
    out_df.Sp .= in_df_W.Sp .+ in_df_E.Sp .+ in_df_S.Sp .+ in_df_N.Sp

    # calculation of aP coefficients
    out_df.aP .= out_df.aW .+ out_df.aE .+ out_df.aS .+ out_df.aN .- out_df.Sp

    # calculation of Su coefficients
    out_df.Su .= in_df_W.Su .+ in_df_E.Su .+ in_df_S.Su .+ in_df_N.Su

    return out_df

end
```

Appendix 1.5

Finite Volume Method Function → “coefficient_matrices()”

```
# generation of A and b matrices using discretised form of a domain

function coefficient_matrices(in_df::DataFrame)

    in_n_cell = in_df.ID[end]
    in_n_div_x = in_df.i[end]
    in_n_div_y = in_df.j[end]

    A=spzeros(Float64,in_n_cell,in_n_cell)

    for ID=1:in_n_cell

        A[ID,ID] = in_df.aP[ID]

        i=rem(ID,in_n_div_x)
        j=div(ID,in_n_div_x)+1
        if i==0
            j=j-1
            i=in_n_div_x
        end

        i_W=i-1
        i_E=i+1
        j_S=j-1
        j_N=j+1

        ID_W=ID-1
        ID_E=ID+1
        ID_S=ID-in_n_div_x
        ID_N=ID+in_n_div_x

        if (i_W > 0)
            A[ID,ID_W] = in_df.aW[ID] *-1
        end

        if (i_E < in_n_div_x+1)
            A[ID,ID_E] = in_df.aE[ID] *-1
        end

        if (j_S > 0)
            A[ID,ID_S] = in_df.aS[ID] *-1
        end

        if (j_N < in_n_div_y+1)
            A[ID,ID_N] = in_df.aN[ID] *-1
        end

    end

    b = in_df.Su

    return A,b

end
```

Appendix 2.1

Intergrid Operation Function → “restrict()”

```
function restrict(in_Cs::DataFrame)

    in_n_div_x = in_Cs.i[end]
    in_n_div_y = in_Cs.j[end]

    out_n_div_x = Int64(in_n_div_x / 2)
    out_n_div_y = Int64(in_n_div_y / 2)
    out_n_cell = Int64(out_n_div_x * out_n_div_y)

    out_Cs=DataFrame(ID=collect(Int64,1:1:out_n_cell))
    out_Cs.i=repeat(collect(Int64,1:1:out_n_div_x),out_n_div_y)
    out_Cs.j=vec(rotl90(repeat(collect(Int64,1:1:out_n_div_y),1,out_n_div_x)))

    out_Cs.x=spzeros(Float64,out_n_cell)
    out_Cs.y=spzeros(Float64,out_n_cell)
    out_Cs.T=spzeros(Float64,out_n_cell)
    out_Cs.r=spzeros(Float64,out_n_cell)

    xs = reshape(in_Cs.x,(in_n_div_x,in_n_div_y))
    ys = reshape(in_Cs.y,(in_n_div_x,in_n_div_y))
    Ts = reshape(in_Cs.T,(in_n_div_x,in_n_div_y))
    rs = reshape(in_Cs.r,(in_n_div_x,in_n_div_y))

    out_Cs[!,:x] .= vec( xs[1:2:in_n_div_x-1,1:2:in_n_div_y-1] .+
xs[2:2:in_n_div_x,1:2:in_n_div_y-1] .+
 xs[1:2:in_n_div_x-1,2:2:in_n_div_y] .+ xs[2:2:in_n_div_x,2:2:in_n_div_y] ) ./ 4
    out_Cs[!,:y] .= vec( ys[1:2:in_n_div_x-1,1:2:in_n_div_y-1] .+
ys[2:2:in_n_div_x,1:2:in_n_div_y-1] .+
 ys[1:2:in_n_div_x-1,2:2:in_n_div_y] .+ ys[2:2:in_n_div_x,2:2:in_n_div_y] ) ./ 4
    out_Cs[!,:T] .= vec( Ts[1:2:in_n_div_x-1,1:2:in_n_div_y-1] .+
Ts[2:2:in_n_div_x,1:2:in_n_div_y-1] .+
 Ts[1:2:in_n_div_x-1,2:2:in_n_div_y] .+ Ts[2:2:in_n_div_x,2:2:in_n_div_y] ) ./ 4
    out_Cs[!,:r] .= vec( rs[1:2:in_n_div_x-1,1:2:in_n_div_y-1] .+
rs[2:2:in_n_div_x,1:2:in_n_div_y-1] .+
 rs[1:2:in_n_div_x-1,2:2:in_n_div_y] .+ rs[2:2:in_n_div_x,2:2:in_n_div_y] ) ./ 4

    return out_Cs

end
```

Appendix 2.2

Intergrid Operation Function → “prolong()”

```
function prolong(in_Cs::DataFrame,bc_df::DataFrame)

    in_n_div_x = in_Cs.i[end]
    in_n_div_y = in_Cs.j[end]

    in_c_dx = (in_Cs.x[end]-in_Cs.x[1])/(in_n_div_x-1)
    in_c_dy = (in_Cs.y[end]-in_Cs.y[1])/(in_n_div_y-1)

    out_n_div_x = Int64( in_n_div_x * 2 )
    out_n_div_y = Int64( in_n_div_y * 2 )
    out_n_cell = Int64( out_n_div_x * out_n_div_y )

    out_c_dx = in_c_dx / 2
    out_c_dy = in_c_dy / 2

    out_Cs=DataFrame(ID=collect(Int64,1:1:out_n_cell))
    out_Cs.i=repeat(collect(Int64,1:1:out_n_div_x),out_n_div_y)
    out_Cs.j=vec(rotl90(repeat(collect(Int64,1:1:out_n_div_y),1,out_n_div_x)))

    out_Cs.x = ( out_Cs.i .* out_c_dx ) .- (out_c_dx/2)
    out_Cs.y = ( out_Cs.j .* out_c_dy ) .- (out_c_dy/2)

    out_Cs.T = spzeros(Float64,out_n_cell)
    out_Cs.r = spzeros(Float64,out_n_cell)

    temp_Cs=face_and_node_values(in_Cs,bc_df)

    Tnn = reshape( out_Cs.T , (out_n_div_x,out_n_div_y))

    Tnn[1:2:out_n_div_x-1,1:2:out_n_div_y-1] .= reshape( ( temp_Cs.T .+ temp_Cs.s_T
.+ temp_Cs.sw_T .+ temp_Cs.w_T ) ./ 4 , (in_n_div_x,in_n_div_y))
    Tnn[2:2:out_n_div_x,1:2:out_n_div_y-1] .= reshape( ( temp_Cs.T .+ temp_Cs.e_T .+
temp_Cs.se_T .+ temp_Cs.s_T ) ./ 4 , (in_n_div_x,in_n_div_y))
    Tnn[1:2:out_n_div_x-1,2:2:out_n_div_y] .= reshape( ( temp_Cs.T .+ temp_Cs.w_T .+
temp_Cs.nw_T .+ temp_Cs.n_T ) ./ 4 , (in_n_div_x,in_n_div_y))
    Tnn[2:2:out_n_div_x,2:2:out_n_div_y] .= reshape( ( temp_Cs.T .+ temp_Cs.n_T .+
temp_Cs.ne_T .+ temp_Cs.e_T ) ./ 4 , (in_n_div_x,in_n_div_y))

    out_Cs.T=vec(Tnn)

    rnn = reshape( out_Cs.r , (out_n_div_x,out_n_div_y))

    rnn[1:2:out_n_div_x-1,1:2:out_n_div_y-1] .= reshape( ( temp_Cs.r .+ temp_Cs.s_r
.+ temp_Cs.sw_r .+ temp_Cs.w_r ) ./ 4 , (in_n_div_x,in_n_div_y))
    rnn[2:2:out_n_div_x,1:2:out_n_div_y-1] .= reshape( ( temp_Cs.r .+ temp_Cs.e_r .+
temp_Cs.se_r .+ temp_Cs.s_r ) ./ 4 , (in_n_div_x,in_n_div_y))
    rnn[1:2:out_n_div_x-1,2:2:out_n_div_y] .= reshape( ( temp_Cs.r .+ temp_Cs.w_r .+
temp_Cs.nw_r .+ temp_Cs.n_r ) ./ 4 , (in_n_div_x,in_n_div_y))
    rnn[2:2:out_n_div_x,2:2:out_n_div_y] .= reshape( ( temp_Cs.r .+ temp_Cs.n_r .+
temp_Cs.ne_r .+ temp_Cs.e_r ) ./ 4 , (in_n_div_x,in_n_div_y))

    out_Cs.r=vec(rnn)

    return out_Cs

end
```

Appendix 2.3

Intergrid Operation Function → “face_and_node_values()”

```
function face_and_node_values(in_Cs::DataFrame,bc_df::DataFrame)

    in_n_cell = in_Cs.ID[end]
    in_n_div_x = in_Cs.i[end]
    in_n_div_y = in_Cs.j[end]

    out_Cs = copy(in_Cs)
    out_n_cell = in_n_cell
    out_n_div_x = in_n_div_x
    out_n_div_y = in_n_div_y

    # residual value assumption at boundaries
    bc_R = 0.0

    # Temperature interpolation

    out_Cs.W_T=spzeros(Float64,in_n_cell)
    out_Cs.E_T=spzeros(Float64,in_n_cell)
    out_Cs.S_T=spzeros(Float64,in_n_cell)
    out_Cs.N_T=spzeros(Float64,in_n_cell)

    for ID=1:out_n_cell

        i=rem(ID,out_n_div_x)
        j=div(ID,out_n_div_x)+1
        if i==0
            j=j-1
            i=out_n_div_x
        end

        if i==1
            out_Cs.W_T[ID]=NaN
        else
            out_Cs.W_T[ID]=in_Cs.T[ID-1]
        end

        if i==out_n_div_x
            out_Cs.E_T[ID]=NaN
        else
            out_Cs.E_T[ID]=in_Cs.T[ID+1]
        end

        if j==1
            out_Cs.S_T[ID]=NaN
        else
            out_Cs.S_T[ID]=in_Cs.T[ID-out_n_div_x]
        end

        if j==out_n_div_y
            out_Cs.N_T[ID]=NaN
        else
            out_Cs.N_T[ID]=in_Cs.T[ID+out_n_div_x]
        end

    end



    out_Cs.SW_T=spzeros(Float64,out_n_cell)
    out_Cs.SE_T=spzeros(Float64,out_n_cell)
    out_Cs.NW_T=spzeros(Float64,out_n_cell)
    out_Cs.NE_T=spzeros(Float64,out_n_cell)

    for ID=1:out_n_cell

        i=rem(ID,out_n_div_x)
        j=div(ID,out_n_div_x)+1
        if i==0
            j=j-1
            i=out_n_div_x
        end

        if i==1 || j==1
            out_Cs.SW_T[ID]=NaN
        else
            out_Cs.SW_T[ID]=in_Cs.T[ID-out_n_div_x-1]
        end
        if i==out_n_div_x || j==1
            out_Cs.SE_T[ID]=NaN
        else
            out_Cs.SE_T[ID]=in_Cs.T[ID-out_n_div_x+1]
        end

        if i==1 || j==out_n_div_y
            out_Cs.NW_T[ID]=NaN
        else
            out_Cs.NW_T[ID]=in_Cs.T[ID+out_n_div_x-1]
        end

        if i==out_n_div_x || j==out_n_div_y
            out_Cs.NE_T[ID]=NaN
        else
            out_Cs.NE_T[ID]=in_Cs.T[ID+out_n_div_x+1]
        end

    end

    # reminder of assumption to corner BC values
    # sw_BC_fix = (bc_df.fix[1]+bc_df.fix[3])/2
    # se_BC_fix = (bc_df.fix[2]+bc_df.fix[3])/2
    # nw_BC_fix = (bc_df.fix[1]+bc_df.fix[4])/2
    # ne_BC_fix = (bc_df.fix[2]+bc_df.fix[4])/2

    out_Cs.sw_T=spzeros(Float64,out_n_cell)
    out_Cs.se_T=spzeros(Float64,out_n_cell)
    out_Cs.nw_T=spzeros(Float64,out_n_cell)
    out_Cs.ne_T=spzeros(Float64,out_n_cell)

    out_Cs[!,:sw_T] .= ifelse.(out_Cs[!,:i].!=1 .&& out_Cs[!,:j].!=1,
(out_Cs[!,:SW_T].+out_Cs[!,:S_T].+out_Cs[!,:W_T].+out_Cs[!,:T])/4, out_Cs[!,:sw_T] )
    out_Cs[!,:sw_T] .= ifelse.(out_Cs[!,:i].==1, bc_df.fix[1], out_Cs[!,:sw_T] )
    out_Cs[!,:sw_T] .= ifelse.(out_Cs[!,:j].==1, bc_df.fix[3], out_Cs[!,:sw_T] )
    out_Cs[!,:sw_T] .= ifelse.(out_Cs[!,:i].==1 .&& out_Cs[!,:j].==1,
(bc_df.fix[1]+bc_df.fix[3])/2, out_Cs[!,:sw_T] )

    out_Cs[!,:se_T] .= ifelse.(out_Cs[!,:i].!=out_n_div_x .&& out_Cs[!,:j].!=1,
(out_Cs[!,:SE_T].+out_Cs[!,:S_T].+out_Cs[!,:E_T].+out_Cs[!,:T])/4, out_Cs[!,:se_T] )
    out_Cs[!,:se_T] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_df.fix[2],
out_Cs[!,:se_T] )
    out_Cs[!,:se_T] .= ifelse.(out_Cs[!,:j].==1, bc_df.fix[3], out_Cs[!,:se_T] )
    out_Cs[!,:se_T] .= ifelse.(out_Cs[!,:i].==out_n_div_x .&& out_Cs[!,:j].==1,
(bc_df.fix[2]+bc_df.fix[3])/2, out_Cs[!,:se_T] )

    out_Cs[!,:nw_T] .= ifelse.(out_Cs[!,:i].!=1 .&& out_Cs[!,:j].!=out_n_div_y,
(out_Cs[!,:NW_T].+out_Cs[!,:N_T].+out_Cs[!,:W_T].+out_Cs[!,:T])/4, out_Cs[!,:nw_T] )
    out_Cs[!,:nw_T] .= ifelse.(out_Cs[!,:i].==1, bc_df.fix[1], out_Cs[!,:nw_T] )
    out_Cs[!,:nw_T] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_df.fix[4],
out_Cs[!,:nw_T] )
    out_Cs[!,:nw_T] .= ifelse.(out_Cs[!,:i].==1 .&& out_Cs[!,:j].==out_n_div_y,
(bc_df.fix[1]+bc_df.fix[4])/2, out_Cs[!,:nw_T] )


    out_Cs[!,:ne_T] .= ifelse.(out_Cs[!,:i].!=out_n_div_x .&&
out_Cs[!,:j].!=out_n_div_y,
(out_Cs[!,:NE_T].+out_Cs[!,:N_T].+out_Cs[!,:E_T].+out_Cs[!,:T])/4, out_Cs[!,:ne_T] )
    out_Cs[!,:ne_T] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_df.fix[2],
out_Cs[!,:ne_T] )
    out_Cs[!,:ne_T] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_df.fix[4],
out_Cs[!,:ne_T] )
    out_Cs[!,:ne_T] .= ifelse.(out_Cs[!,:i].==out_n_div_x .&&
out_Cs[!,:j].==out_n_div_y, (bc_df.fix[2]+bc_df.fix[4])/2, out_Cs[!,:ne_T] )

    out_Cs[!,:w_T] .= ifelse.(out_Cs[!,:i].==1, bc_df.fix[1],
(out_Cs[!,:W_T].+out_Cs[!,:T].+out_Cs[!,:nw_T].+out_Cs[!,:sw_T])/4)
    out_Cs[!,:e_T] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_df.fix[2],
(out_Cs[!,:E_T].+out_Cs[!,:T].+out_Cs[!,:ne_T].+out_Cs[!,:se_T])/4)
    out_Cs[!,:s_T] .= ifelse.(out_Cs[!,:j].==1, bc_df.fix[3],
(out_Cs[!,:S_T].+out_Cs[!,:T].+out_Cs[!,:sw_T].+out_Cs[!,:se_T])/4)
    out_Cs[!,:n_T] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_df.fix[4],
(out_Cs[!,:N_T].+out_Cs[!,:T].+out_Cs[!,:nw_T].+out_Cs[!,:ne_T])/4)



    # residual interpolation

    out_Cs.W_r=spzeros(Float64,in_n_cell)
    out_Cs.E_r=spzeros(Float64,in_n_cell)
    out_Cs.S_r=spzeros(Float64,in_n_cell)
    out_Cs.N_r=spzeros(Float64,in_n_cell)

    for ID=1:out_n_cell
```

```
        i=rem(ID,out_n_div_x)
        j=div(ID,out_n_div_x)+1
        if i==0
            j=j-1
            i=out_n_div_x
        end

        if i==1
            out_Cs.W_r[ID]=NaN
        else
            out_Cs.W_r[ID]=in_Cs.r[ID-1]
        end

        if i==out_n_div_x
            out_Cs.E_r[ID]=NaN
        else
            out_Cs.E_r[ID]=in_Cs.r[ID+1]
        end

        if j==1
            out_Cs.S_r[ID]=NaN
        else
            out_Cs.S_r[ID]=in_Cs.r[ID-out_n_div_x]
        end

        if j==out_n_div_y
            out_Cs.N_r[ID]=NaN
        else
            out_Cs.N_r[ID]=in_Cs.r[ID+out_n_div_x]
        end

    end

    out_Cs.SW_r=spzeros(Float64,out_n_cell)
    out_Cs.SE_r=spzeros(Float64,out_n_cell)
    out_Cs.NW_r=spzeros(Float64,out_n_cell)
    out_Cs.NE_r=spzeros(Float64,out_n_cell)

    for ID=1:out_n_cell

        i=rem(ID,out_n_div_x)
        j=div(ID,out_n_div_x)+1
        if i==0
            j=j-1
            i=out_n_div_x
        end

        if i==1 || j==1
            out_Cs.SW_r[ID]=NaN
        else
            out_Cs.SW_r[ID]=in_Cs.r[ID-out_n_div_x-1]
        end

        if i==out_n_div_x || j==1
            out_Cs.SE_r[ID]=NaN
        else
            out_Cs.SE_r[ID]=in_Cs.r[ID-out_n_div_x+1]
        end

        if i==1 || j==out_n_div_y
            out_Cs.NW_r[ID]=NaN
        else
            out_Cs.NW_r[ID]=in_Cs.r[ID+out_n_div_x-1]
        end

        if i==out_n_div_x || j==out_n_div_y
            out_Cs.NE_r[ID]=NaN
        else
            out_Cs.NE_r[ID]=in_Cs.r[ID+out_n_div_x+1]
        end

    end

    out_Cs.sw_r=spzeros(Float64,out_n_cell)
    out_Cs.se_r=spzeros(Float64,out_n_cell)
    out_Cs.nw_r=spzeros(Float64,out_n_cell)
    out_Cs.ne_r=spzeros(Float64,out_n_cell)

    out_Cs[!,:sw_r] .= ifelse.(out_Cs[!,:i].!=1 .&& out_Cs[!,:j].!=1,
(out_Cs[!,:SW_r].+out_Cs[!,:S_r].+out_Cs[!,:W_r].+out_Cs[!,:r])/4, out_Cs[!,:sw_r] )
    out_Cs[!,:sw_r] .= ifelse.(out_Cs[!,:i].==1, bc_R, out_Cs[!,:sw_r] )
    out_Cs[!,:sw_r] .= ifelse.(out_Cs[!,:j].==1, bc_R, out_Cs[!,:sw_r] )
    out_Cs[!,:sw_r] .= ifelse.(out_Cs[!,:i].==1 .&& out_Cs[!,:j].==1, bc_R,
out_Cs[!,:sw_r] )
```

```
    out_Cs[!,:se_r] .= ifelse.(out_Cs[!,:i].!=out_n_div_x .&& out_Cs[!,:j].!=1,
(out_Cs[!,:SE_r].+out_Cs[!,:S_r].+out_Cs[!,:E_r].+out_Cs[!,:r])/4, out_Cs[!,:se_r] )
    out_Cs[!,:se_r] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_R, out_Cs[!,:se_r] )
    out_Cs[!,:se_r] .= ifelse.(out_Cs[!,:j].==1, bc_R, out_Cs[!,:se_r] )
    out_Cs[!,:se_r] .= ifelse.(out_Cs[!,:i].==out_n_div_x .&& out_Cs[!,:j].==1, bc_R,
out_Cs[!,:se_r] )

    out_Cs[!,:nw_r] .= ifelse.(out_Cs[!,:i].!=1 .&& out_Cs[!,:j].!=out_n_div_y,
(out_Cs[!,:NW_r].+out_Cs[!,:N_r].+out_Cs[!,:W_r].+out_Cs[!,:r])/4, out_Cs[!,:nw_r] )
    out_Cs[!,:nw_r] .= ifelse.(out_Cs[!,:i].==1, bc_R, out_Cs[!,:nw_r] )
    out_Cs[!,:nw_r] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_R, out_Cs[!,:nw_r] )
    out_Cs[!,:nw_r] .= ifelse.(out_Cs[!,:i].==1 .&& out_Cs[!,:j].==out_n_div_y, bc_R,
out_Cs[!,:nw_r] )

    out_Cs[!,:ne_r] .= ifelse.(out_Cs[!,:i].!=out_n_div_x .&&
out_Cs[!,:j].!=out_n_div_y,
(out_Cs[!,:NE_r].+out_Cs[!,:N_r].+out_Cs[!,:E_r].+out_Cs[!,:r])/4, out_Cs[!,:ne_r] )
    out_Cs[!,:ne_r] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_R, out_Cs[!,:ne_r] )
    out_Cs[!,:ne_r] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_R, out_Cs[!,:ne_r] )
    out_Cs[!,:ne_r] .= ifelse.(out_Cs[!,:i].==out_n_div_x .&&
out_Cs[!,:j].==out_n_div_y, bc_R, out_Cs[!,:ne_r] )

    out_Cs[!,:w_r] .= ifelse.(out_Cs[!,:i].==1, bc_R,
(out_Cs[!,:W_r].+out_Cs[!,:r].+out_Cs[!,:nw_r].+out_Cs[!,:sw_r])/4)
    out_Cs[!,:e_r] .= ifelse.(out_Cs[!,:i].==out_n_div_x, bc_R,
(out_Cs[!,:E_r].+out_Cs[!,:r].+out_Cs[!,:ne_r].+out_Cs[!,:se_r])/4)
    out_Cs[!,:s_r] .= ifelse.(out_Cs[!,:j].==1, bc_R,
(out_Cs[!,:S_r].+out_Cs[!,:r].+out_Cs[!,:sw_r].+out_Cs[!,:se_r])/4)
    out_Cs[!,:n_r] .= ifelse.(out_Cs[!,:j].==out_n_div_y, bc_R,
(out_Cs[!,:N_r].+out_Cs[!,:r].+out_Cs[!,:nw_r].+out_Cs[!,:ne_r])/4)

    return out_Cs

end
```

Appendix 3.1

Iterative Solution Function → “gauss_seidel_iterate()”

```
# iterator function, performs one iteration of gauss seidel
# uses sparse coefficient matrice, visits all rows while visiting only columns with
nonzero elements

function gauss_seidel_iterate(A,b,x)

    sA = size(x,1)
    row_sum = spzeros(Float64,sA) # initializes sparse array

    for i=1:sA

        columns = findall(!iszero,A[i,:]) # finds nonzero column indices

        for j in columns
            row_sum[i]=row_sum[i]+A[i,j]*x[j]
        end

        x[i] = ( b[i] - ( row_sum[i] - A[i,i]*x[i] ) ) / A[i,i]

    end

    return x

end
```

Appendix 3.2

Iterative Solution Function → “iterate_solution_count()”

```
# iterative solution of A.x=b using predefined conditioning
# ic = allowed iteration count
# i = current iteration count

function iterate_solution_count(A,b,x,ic)

    R=Float64[]
    S=Float64[]
    t=Int64[]
    r=Float64[]

    y = copy(x).+1
    i=1

    rh=ones(Float64,size(x))
    xh=copy(x)

    st=time_ns()
    while i<=ic
        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]
        i=i+1
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)#:auto)

    return iteration_history,field_history,residual_history

end
```

Appendix 3.3

Iterative Solution Function → “iterate_solution_count_for_cycles()”

```
# iterative solution of A.x=b using predefined conditioning
# ic = allowed iteration count
# i = current iteration count

function iterate_solution_count_for_cycles(A,b,x,r,ic)

    R=Float64[]
    S=Float64[]
    t=Int64[]

    y = copy(x).+r
    i=1

    rh=ones(Float64,size(x))
    xh=copy(x)

    st=time_ns()
    while i<=ic
        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]
        i=i+1
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)#:auto)

    return iteration_history,field_history,residual_history

end
```

Appendix 3.4

Iterative Solution Function → “iterate_solution_converge_for_cycles()”

```
# iterative solution of A.x=b using predefined conditioning
# dc = desired convergence
# mean(r) = mean residual

function iterate_solution_converge_for_cycles(A,b,x,r,dc)

    R=Float64[]
    S=Float64[]
    t=Int64[]

    y = copy(x).+r

    rh=ones(Float64,size(x))
    xh=copy(x)

    print("mean(r)=",mean(r))
    print("dc=",dc)

    st=time_ns()
    while mean(r)>dc
        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)#:auto)

    return iteration_history,field_history,residual_history

end
```

Appendix 3.5

Iterative Solution Function → “iterate_solution_converge_with_count()”

```
function iterate_solution_converge_with_count(A,b,x,cc,i_max)

    R=Float64[]
    S=Float64[]
    t=Int64[]
    r=Float64[]

    y = copy(x).+1
    cR=1
    ic=1

    rh=ones(Float64,size(x))
    xh=copy(x)

    st=time_ns()
    ct=st
    while cR>cc && ic<=i_max
        pt=time_ns()-st
        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]
        cR=R[end]
        ic=ic+1
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)

    return iteration_history,field_history,residual_history

end
```

Appendix 3.6

Iterative Solution Function → “iterate_solution_flexible()”

```
# pc = prolong criteria
# rc = restrict criteria
# fc = final convergence

function iterate_solution_flexible(A,b,x,r,pc,rc,i_max,i_min,fc)

    R=Float64[]
    S=Float64[]
    t=Int64[]

    cR=mean(r)
    pR=mean(r)
    y = copy(x) .+ r
    ic=1

    rh=ones(Float64,size(x))
    xh=copy(x)

    println("iterations")
    println("initial conditions")
    println(">ic=",ic,">pc=",pc,">pR=",pR,">cR=",cR,">cR/pR=",cR/pR,">")
    println("start")

    st=time_ns()
    while ic<=i_max

        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]

        if size(R,1)>1
            pR = R[end-1]
        end
        cR = R[end]
        println(">ic=",ic,">pc=",pc,">pR=",pR,">cR=",cR,">cR/pR=",cR/pR,">")
        ic=ic+1

        if cR<fc && ic>i_min
            action="prolong"
            break
        elseif cR<pc && ic>i_min
            action="prolong"
            break
        elseif rc<(cR/pR) && ic>i_min
            action="restrict"
            break
        end

    end

    if ic>i_max
        action="prolong"
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)#:auto)

    return iteration_history,field_history,residual_history,action

end
```

Appendix 3.7

Iterative Solution Function → “iterate_solution_PID_driven()”

```
# cP, cI, cD : coefficients of P,I,D
# Pc, Ic, Dc : criteria of P,I,D


function iterate_solution_PID_driven(A,b,x,r,cP,cI,cD,Pc,Ic,Dc,i_min,i_max,fc)

    R=Float64[]
    S=Float64[]
    t=Int64[]

    cR=mean(r)
    pR=mean(r)
    iR=mean(r)

    y = copy(x) .+ r
    ic=1
    integral=0

    rh=ones(Float64,size(x))
    xh=copy(x)

    println("ic : iteration count , R : mean residual , pR : previous mean residual ,
cR : current mean residual")

    st=time_ns()
    while ic<=i_max

        x=gauss_seidel_iterate(A,b,x)
        print(">") # to get a feeling if its running
        r = abs.((x.-y))
        y.=x
        ct=time_ns()-st
        push!(R,mean(r))
        push!(S,std(r))
        push!(t,ct)
        rh = [rh r]
        xh = [xh x]

        if size(R,1)>1
            pR = R[end-1]
        end
        cR = R[end]
        if cR>iR
            iR=cR
        end

        ic=ic+1

        # calculation of PID terms
        previous_error = pR - fc
        current_error = cR - fc
        integral = integral + (current_error+previous_error)/2
        change = current_error - previous_error
        proportional_term = cP * current_error
        integral_term = cI * integral
        derivative_term = cD * change

        println(@sprintf ">ic=%d\t>pR=%.6f\t>cR=%.6f\t>cR/pR=%.6f\t>cR-
pR=%.6f\t>sum(R)=%.6f\t>iR=%.6f\t>P=%.6f\t>I=%.6f\t>D=%.6f"
  ic-1  pR  cR  cR/pR  cR-pR  sum(R)  iR  proportional_term  integral_term
derivative_term )

        if proportional_term<Pc*iR && ic>i_min
            action="prolong"
            println("proportional criteria triggered.")
            break
        elseif integral_term>Ic*iR && ic>i_min
            action="prolong"
            println("integral criteria triggered.")
            break
        elseif derivative_term>Dc*iR && ic>i_min
            action="restrict"
            println("derivative criteria triggered.")
            break
        end

    end
    if ic>i_max
        action="prolong"
        println("maximum number of iterations criteria triggered.")
    end

    iteration_history=DataFrame(R=R[1:end],S=S[1:end],t=t[1:end])
    field_history=DataFrame(xh[:,2:end],:auto)
    residual_history=DataFrame(rh[:,2:end],:auto)#:auto)

    return iteration_history,field_history,residual_history,action

end
```

Appendix 4.1

Multigrid Algorithm Function → “preprocess_and_initialize()”

```
function
preprocess_and_initialize(dim1::Float64,dim2::Float64,nx::Int64,ny::Int64,bc_df::Data
Frame,cc::Float64,ic::Int64,init_val::Float64)

    n_cell=nx*ny


data=DataFrame(step=Int64[],level=Int64[],n_cell=Int64[],nx=Int64[],ny=Int64[],initia
l_residual=Float64[],final_residual=Float64[],
iteration_count=Int64[],iteration_duration=Int64[],operation_duration=Int64[],FVM_dur
ation=Int64[],initial_field=DataFrame[],
final_field=DataFrame[],iteration_history=DataFrame[],field_history=DataFrame[],resid
ual_history=DataFrame[])

    # time stamp for operation start
    op_st=time_ns()
    # calling FVM functions
    Cs_i = initialize_domain(dim1,dim2,nx,ny)
    Cs_i.r = ones(Float64,n_cell)
    # operation duration
    op_dur=time_ns()-op_st

    # time stamp for FVM functions
    fvm_st=time_ns()
    # calling FVM functions
    Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
    Cs_DF=discretised_form(Cs_BC)
    A,b=coefficient_matrices(Cs_DF)
    # FVM function duration
    fvm_dur=time_ns()-fvm_st

    x = ones(Float64,n_cell).*init_val

    st=time_ns()
    if ic>0

iteration_history,field_history,residual_history=iterate_solution_converge_with_count
(A,b,x,cc,ic)
    elseif ic<1
        ct=time_ns()-st
        iteration_history=DataFrame(R=[mean(Cs_i.r)],S=[std(Cs_i.r)],t=[ct])
        field_history=DataFrame([Cs_i.T],:auto)
        residual_history=DataFrame([Cs_i.r],:auto)
    end

    # assigning solution values to initial domain
    Cs_f=copy(Cs_i)
    Cs_f.T.=x
    Cs_f.r=residual_history[:,end]

push!(data,(1,0,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],size(ite
ration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

    return data

end
```

Appendix 4.2

Multigrid Algorithm Function → “run_fixed_V_cycle()”

```
function
run_fixed_V_cycle(preprocess::DataFrame,bc_df::DataFrame,course_level::Int64,i_sweep:
:Int64)

    data=copy(preprocess)

    max_level=find_max_level(data.nx[end],data.ny[end])

    if course_level>max_level
        course_level=max_level
        println("desired course level is infeasible, max course level is set to : ",
course_level)
    else
        println("desired course level is feasible, max course level is set to : ",
course_level)
    end

    levels = [Vector(data.level[end]:1:course_level);Vector(course_level-1:-1:0)]
    println("fixed V cycle levels will be :", levels)

    for level in levels

        # time stamp intergrid operation start
        op_st = time_ns()

        if level>data.level[end]
            Cs_i=restrict(data.final_field[end])
        elseif level<data.level[end]
            Cs_i=prolong(data.final_field[end],bc_df)
        else
            Cs_i=data.final_field[end]
        end

        # intergrid operation duration
        op_dur = time_ns()-op_st

        # assigning step
        step=data.step[end]+1
        print("step=",step," level=",level," iterations...")

        # assigning field info
        n_cell=Cs_i.ID[end]
        nx=Cs_i.i[end]
        ny=Cs_i.j[end]

        # time stamp for FVM functions
        fvm_st=time_ns()
        # calling FVM functions
        Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
        Cs_DF=discretised_form(Cs_BC)
        A,b=coefficient_matrices(Cs_DF)
        # FVM function duration
        fvm_dur=time_ns()-fvm_st

        # iterative solution
        x=copy(Cs_i.T)
        r=copy(Cs_i.r)

iteration_history,field_history,residual_history=iterate_solution_count_for_cycles(A,
b,x,r,i_sweep)

        # assigning solution values to final field
        Cs_f=copy(Cs_i)
        Cs_f.T .= x
        Cs_f.r .= residual_history[:,end]

push!(data,(step,level,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],s
ize(iteration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

    end

    return data

end
```

Appendix 4.3

Multigrid Algorithm Function → “run_fixed_W_cycle()”

```
function
run_fixed_W_cycle(preprocess::DataFrame,bc_df::DataFrame,course_level::Int64,i_sweep:
:Int64)

    data=copy(preprocess)

    max_level=find_max_level(data.nx[end],data.ny[end])

    if course_level>max_level
        course_level=max_level
        println("desired course level is infeasible, max course level is set to : ",
course_level)
    else
        println("desired course level is feasible, max course level is set to : ",
course_level)
    end

    levels = [Vector(data.level[end]:1:course_level);course_level-
1;Vector(course_level:-1:1);Vector(2:1:course_level);course_level-
1;Vector(course_level:-1:0)]
    println("fixed W cycle levels will be :", levels)

    for level in levels

        # time stamp intergrid operation start
        op_st = time_ns()

        if level>data.level[end]
            Cs_i=restrict(data.final_field[end])
        elseif level<data.level[end]
            Cs_i=prolong(data.final_field[end],bc_df)
        else
            Cs_i=data.final_field[end]
        end

        # intergrid operation duration
        op_dur = time_ns()-op_st
        # assigning step
        step=data.step[end]+1
        print("step=",step," level=",level," iterations...")
        # assigning field info
        n_cell=Cs_i.ID[end]
        nx=Cs_i.i[end]
        ny=Cs_i.j[end]

        # time stamp for FVM functions
        fvm_st=time_ns()
        # calling FVM functions
        Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
        Cs_DF=discretised_form(Cs_BC)
        A,b=coefficient_matrices(Cs_DF)
        # FVM function duration
        fvm_dur=time_ns()-fvm_st

        # iterative solution
        x=copy(Cs_i.T)
        r=copy(Cs_i.r)

iteration_history,field_history,residual_history=iterate_solution_count_for_cycles(A,
b,x,r,i_sweep)

        # assigning solution values to final field
        Cs_f=copy(Cs_i)
        Cs_f.T .= x
        Cs_f.r .= residual_history[:,end]

push!(data,(step,level,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],s
ize(iteration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

    end

    return data

end
```

Appendix 4.4

Multigrid Algorithm Function → "run_fixed_F_cycle()"

```
function
run_fixed_F_cycle(preprocess::DataFrame,bc_df::DataFrame,course_level::Int64,i_sweep:
:Int64)

    data=copy(preprocess)

    max_level=find_max_level(data.nx[end],data.ny[end])

    if course_level>max_level
        course_level=max_level
        println("desired course level is infeasible, max course level is set to : ",
course_level)
    else
        println("desired course level is feasible, max course level is set to : ",
course_level)
    end

    temp_course_levels = [ [Vector(course_level:-
1:min_level);Vector(min_level+1:1:course_level-1)] for min_level=course_level-1:-1:1
]

    course_levels=Int64[]

    for i in eachindex(temp_course_levels)

        for j in eachindex(temp_course_levels[i])

        push!(course_levels, temp_course_levels[i][j])

        end

    end

    levels = [Vector(data.level[end]:1:course_level-
1);course_levels;Vector(course_level:-1:0)]
    println("fixed F cycle levels will be :", levels)

    for level in levels

        # time stamp intergrid operation start
        op_st = time_ns()

        if level>data.level[end]
            Cs_i=restrict(data.final_field[end])
        elseif level<data.level[end]
            Cs_i=prolong(data.final_field[end],bc_df)
        else
            Cs_i=data.final_field[end]
        end

        # intergrid operation duration
        op_dur = time_ns()-op_st

        # assigning step
        step=data.step[end]+1

        print("step=",step," level=",level," iterations...")

        # assigning field info
        n_cell=Cs_i.ID[end]
        nx=Cs_i.i[end]
        ny=Cs_i.j[end]

        # time stamp for FVM functions
        fvm_st=time_ns()
        # calling FVM functions
        Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
        Cs_DF=discretised_form(Cs_BC)
        A,b=coefficient_matrices(Cs_DF)
        # FVM function duration
        fvm_dur=time_ns()-fvm_st

        # iterative solution
        x=copy(Cs_i.T)
        r=copy(Cs_i.r)
```

```
iteration_history,field_history,residual_history=iterate_solution_count_for_cycles(A,
b,x,r,i_sweep)

        # assigning solution values to final field
        Cs_f=copy(Cs_i)
        Cs_f.T .= x
        Cs_f.r .= residual_history[:,end]

        # adding data

push!(data,(step,level,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],s
ize(iteration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

    end

    return data

end
```

Appendix 4.5

Multigrid Algorithm Function → “run_flexible_cycle()”

```
function
run_flexible_cycle(preprocess_data::DataFrame,bc_df::DataFrame,alpha::Float64,beta::F
loat64,
fc::Float64,i_min::Int64,i_max::Int64,step_max::Int64)

    decision=Dict("restrict"=>1,"prolong"=>-1,"continue"=>0)

    # time stamp intergrid operation start
    op_st = time_ns()

    data=copy(preprocess_data)

    Cs_i=data.final_field[end]
    action="continue"

    # intergrid operation duration
    op_dur = time_ns()-op_st

    condition = false
    while condition == false

        # assigning previous level info
        previous_step=data.step[end]
        previous_level=data.level[end]
        previous_residual=data.iteration_history[end].R[end]
        println(">",previous_residual,">")

        # recognizing previous action and calculating current level
        level=previous_level+decision[action]
        println(">",level,">")
        step=previous_step+1
        println(">",step,">")

        # assigning field info
        n_cell=Cs_i.ID[end]
        nx=Cs_i.i[end]
        ny=Cs_i.j[end]

        # time stamp for FVM functions
        fvm_st=time_ns()
        # calling FVM functions
        Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
        Cs_DF=discretised_form(Cs_BC)
        A,b=coefficient_matrices(Cs_DF)

        # FVM function duration
        fvm_dur=time_ns()-fvm_st

        # iterative solution
        x=copy(Cs_i.T)
        r=Cs_i.r
        pc=previous_residual*alpha
        if pc<fc
            pc=fc
            println("!!! PROLONGATION CRITERIA RESET !!!")
        end
        rc=beta
        if previous_residual<fc
            rc=1
            println("!!! RESTRICTION CRITERIA RESET !!!")
        end

        println("pre_res=",previous_residual,",","mean(r)=",mean(r))

iteration_history,field_history,residual_history,action=iterate_solution_flexible(A,b
,x,r,pc,rc,i_max,i_min,fc)
        println(">",action,">")

        # assigning solution values to initial domain
        Cs_f=copy(Cs_i)
        Cs_f.T.=x
        Cs_f.r.=residual_history[:,end]

        # adding data

push!(data,(step,level,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],s
ize(iteration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

        # determining feasiblity and next level starting conditions
        restrict_infeasible = isodd(nx) || isodd(ny)
        next_level=level+decision[action]

        # time stamp intergrid operation start
        op_st = time_ns()

        if level==0 && iteration_history.R[end]<fc
            condition = true
        elseif level==0 && next_level<level
            Cs_i=copy(Cs_f)
            action="continue"
            println(">can not exceed maximum resolution level, thus action is changed
to=>",action,">")
            condition = false
        elseif level>0 && iteration_history.R[end]<fc
            Cs_i=prolong(Cs_f,bc_df)
            action="prolong"
            println(">final convergence is reached, thus action is changed
to=>",action,">")
            condition = false
        elseif next_level>level && restrict_infeasible==true
            Cs_i=copy(Cs_f)
            action="continue"
            println(">restrict is infeasible, thus action is changed
to=>",action,">")
            condition = false
        elseif next_level>level && restrict_infeasible==false
            Cs_i=restrict(Cs_f)
            condition = false
        elseif level>0 && next_level<level
            Cs_i=prolong(Cs_f,bc_df)
            condition = false
        else
            Cs_i=copy(Cs_f)
            action="continue"
            condition=false
        end

        # intergrid operation duration
        op_dur = time_ns()-op_st

        if step>step_max
            break
        end

    end

    return data

end
```

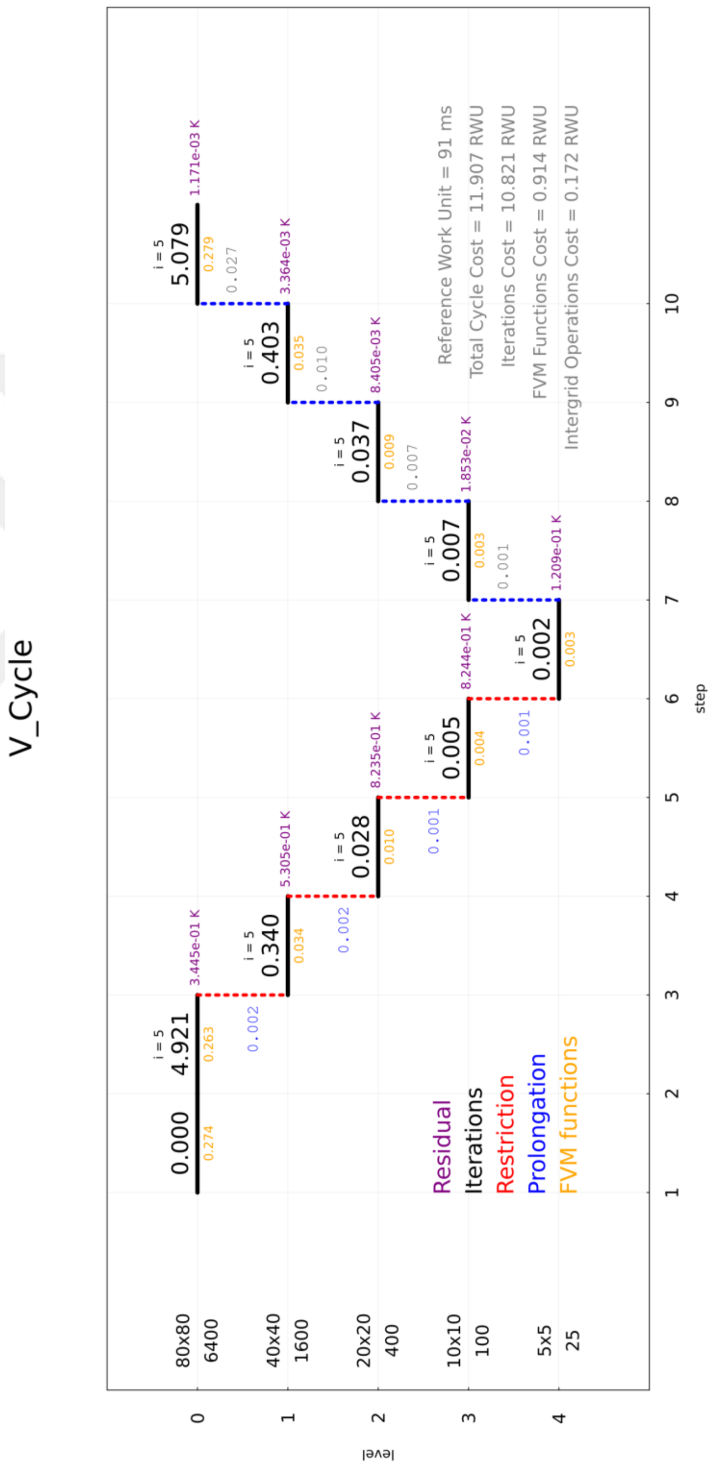
Appendix 4.6

Multigrid Algorithm Function → “run_PID_driven_cycle()”

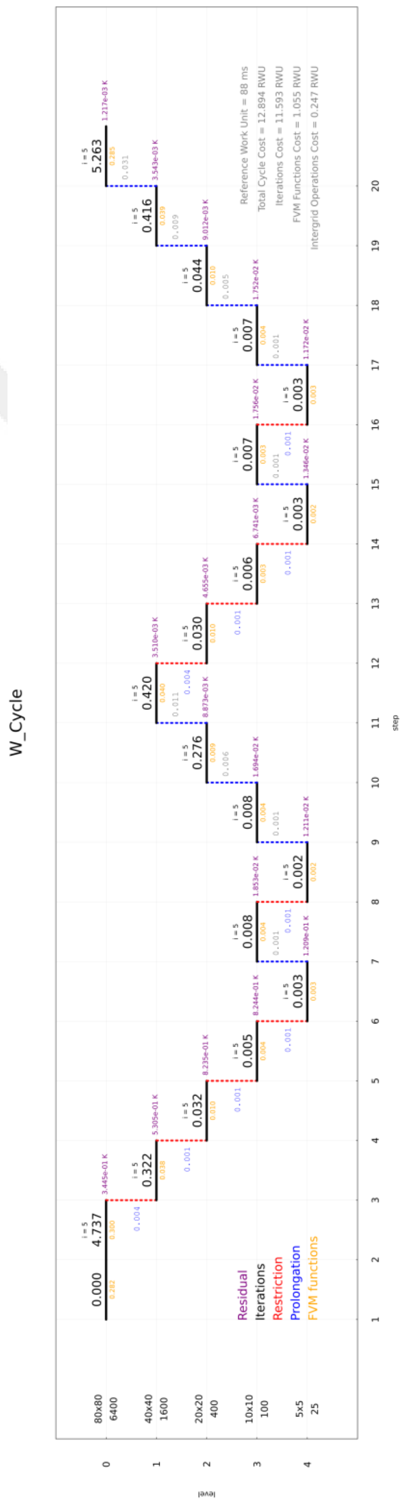
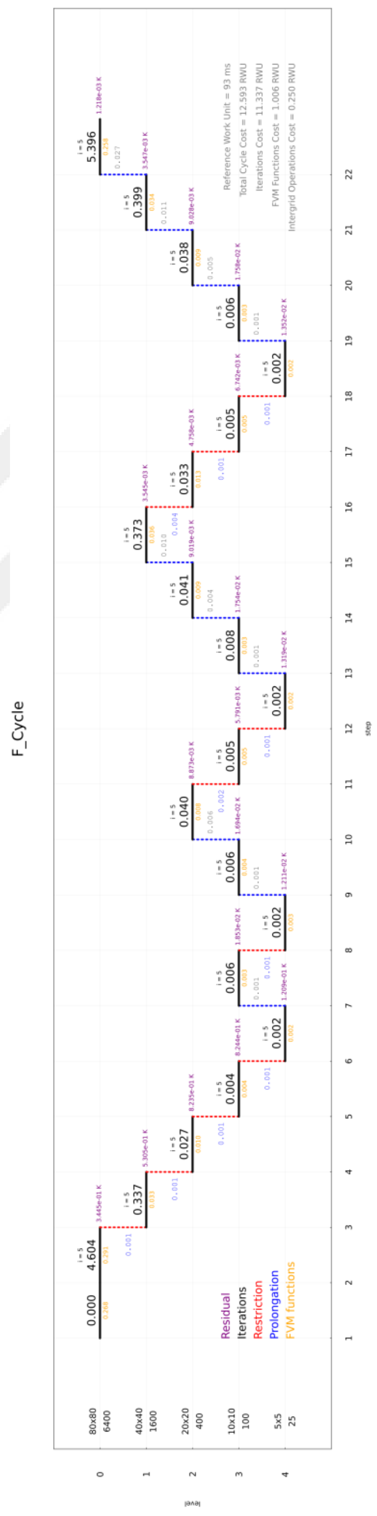
```
# cP : coefficient of proportional term
# cI : coefficient of integral term
# cD : coefficient of derivative term
# Pc : criteria of of proportional term
# Ic : criteria of integral term
# Dc : criteria of derivative term
# fc : final convergence
# i_min : minimum number of iterations for a resolution
# i_max : maximum number of iterations for a resolution
# step_max : maximum number of steps for the run


function run_PID_driven(preprocess_data::DataFrame,boundary_conditions::DataFrame,
PID_coefficients::Vector,PID_criteria::Vector,operation_limits::Vector)

    decision=Dict("restrict"=>1,"prolong"=>-1,"continue"=>0)

    println("PID driven multigrid cycle.")

    println("reading input..")

    cP = PID_coefficients[1]
    cI = PID_coefficients[2]
    cD = PID_coefficients[3]

    Pc = PID_criteria[1]
    Ic = PID_criteria[2]
    Dc = PID_criteria[3]

    fc = operation_limits[1]
    i_min = operation_limits[2]
    i_max = operation_limits[3]
    step_max = operation_limits[4]

    # time stamp intergrid operation start
    op_st = time_ns()

    data=copy(preprocess_data)
    bc_df=copy(boundary_conditions)

    Cs_i=data.final_field[end]
    action="continue"

    # intergrid operation duration
    op_dur = time_ns()-op_st

    println("initiating cycle...")

    progress = true
    while progress

        # assigning previous level info
        previous_step=data.step[end]
        previous_level=data.level[end]
        previous_residual=data.iteration_history[end].R[end]

        # recognizing previous action and calculating current level
        level=previous_level+decision[action]
        println("level = ",level)
        step=previous_step+1
        println("step = ",step)

        # assigning field info
        n_cell=Cs_i.ID[end]
        nx=Cs_i.i[end]
        ny=Cs_i.j[end]
        # determining restriction feasibility
        restrict_infeasible = isodd(nx) || isodd(ny)

        # time stamp for FVM functions
        fvm_st=time_ns()
        # calling FVM functions
        Cs_BC=apply_boundary_conditions(Cs_i,bc_df)
        Cs_DF=discretised_form(Cs_BC)
        A,b=coefficient_matrices(Cs_DF)
        # FVM function duration
        fvm_dur=time_ns()-fvm_st
        # iterative solution
        x=copy(Cs_i.T)
        r=Cs_i.r
        println("residual of previous step = ",previous_residual," , ","residual of
current step= ",mean(r))

        if level<1
            Pc=0.0
            println("PROPORTIONAL CRITERIA RESET")
            println(@sprintf ">Pc=%.1f" Pc)
        else
            Pc = PID_criteria[1]
        end


        if restrict_infeasible
            Ic=1/fc
            println("INTEGRAL CRITERIA RESET")
            println(@sprintf ">Ic=%.1f" Ic)
        else
            Ic = PID_criteria[2]
        end

        if previous_residual<fc
            Dc = 0.0
            println("DERIVATIVE CRITERIA RESET")
            println(@sprintf ">Dc=%.1f" Dc)
        else
            Dc = PID_criteria[3]
        end


iteration_history,field_history,residual_history,action=iterate_solution_PID_driven(A
,b,x,r,cP,cI,cD,Pc,Ic,Dc,i_min,i_max,fc)
        if level>0
            println("action = ",action)
        end

        # assigning solution values to initial domain
        Cs_f=copy(Cs_i)
        Cs_f.T.=x
        Cs_f.r.=residual_history[:,end]

        # adding data

push!(data,(step,level,n_cell,nx,ny,iteration_history.R[1],iteration_history.R[end],s
ize(iteration_history,1),
iteration_history.t[end],op_dur,fvm_dur,Cs_i,Cs_f,iteration_history,field_history,res
idual_history))

        # determining next level
        next_level=level+decision[action]

        # time stamp intergrid operation start
        op_st = time_ns()


        if level==0 && iteration_history.R[end]<fc
            progress = false
        elseif level==0 && next_level<level
            Cs_i=copy(Cs_f)
            action="continue"
            println("can not exceed maximum resolution level, thus action is changed
to => ",action)
            progress = true
        elseif level>0 && iteration_history.R[end]<fc
            Cs_i=prolong(Cs_f,bc_df)
            action="prolong"
            println("final convergence is reached, thus action is changed to =>
",action)
            progress = true
        elseif next_level>level && restrict_infeasible==true
            Cs_i=copy(Cs_f)
            action="continue"
            println(">restrict is infeasible, thus action is changed
to=>",action,">")
            progress = true
        elseif next_level>level && restrict_infeasible==false
            Cs_i=restrict(Cs_f)
            progress = true
        elseif level>0 && next_level<level
            Cs_i=prolong(Cs_f,bc_df)
            progress = true
        else
            Cs_i=copy(Cs_f)
            action="continue"
            progress=true
        end

        # intergrid operation duration
        op_dur = time_ns()-op_st

        if data.step[end]>step_max
            progress = false
        end

    end

    println("PID driven multigrid cycle completed.")
    println("")

    return data

end
```

Appendix 5.1

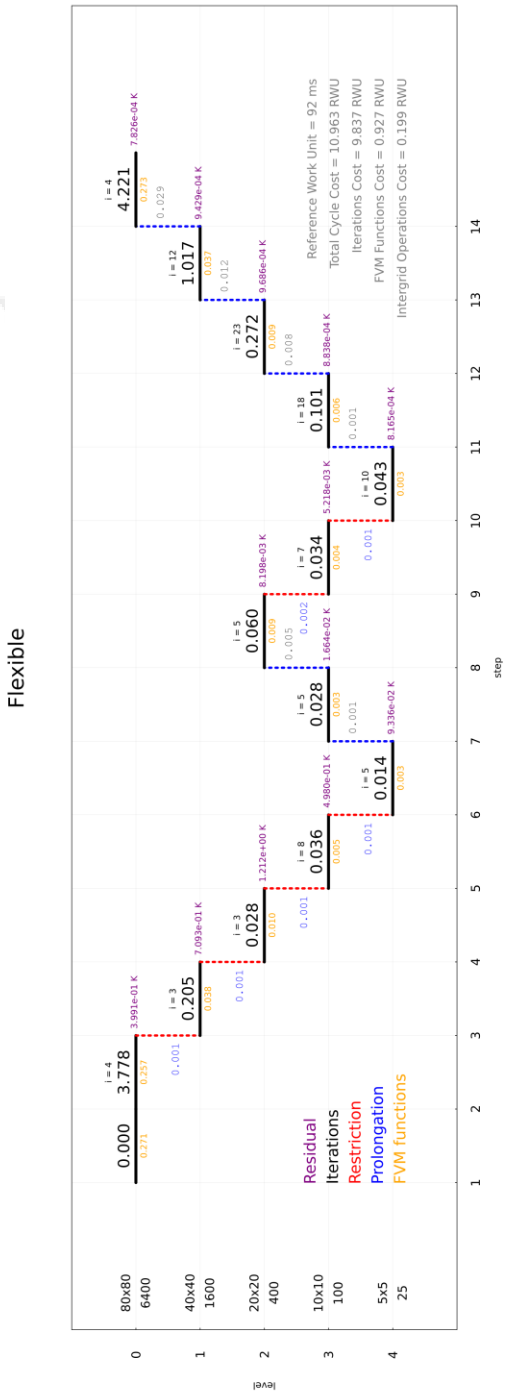
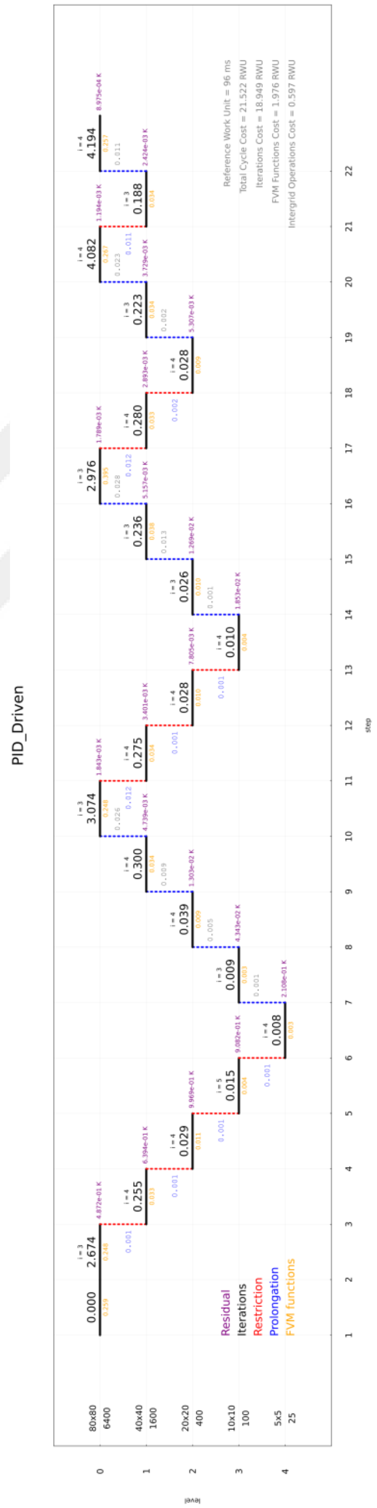
Validation Run Summary Plot of V Cycle



Appendix 5.2

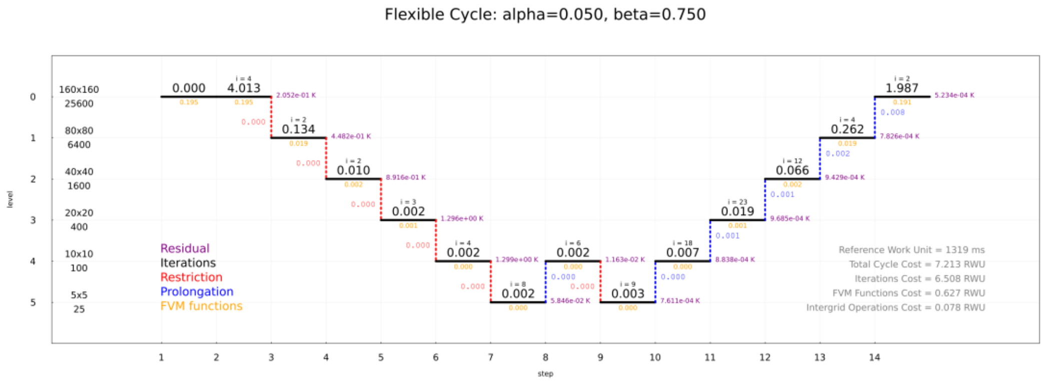
Validation Run Summary Plot of W Cycle	Validation Run Summary Plot of F Cycle
	

Appendix 5.3

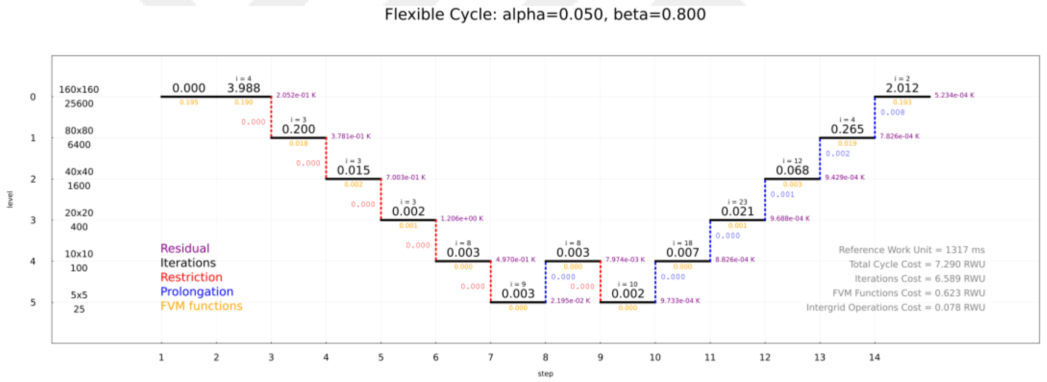
Validation Run Summary Plot of Flexible Cycle	Validation Run Summary Plot of PID Driven Cycle
	

Appendix 6.1

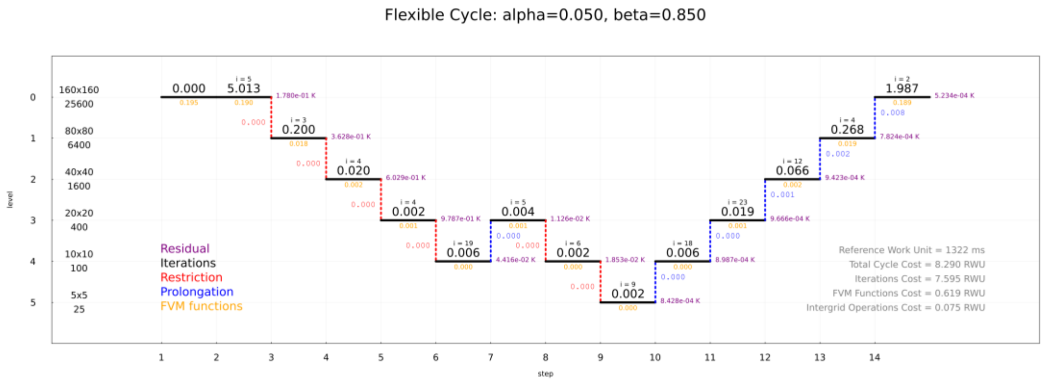
Run Summary Plot of Flexible Cycle → Alpha=0.05 Beta=0.75 → 7.21 RWU



Run Summary Plot of Flexible Cycle → Alpha=0.05 Beta=0.80 → 7.29 RWU

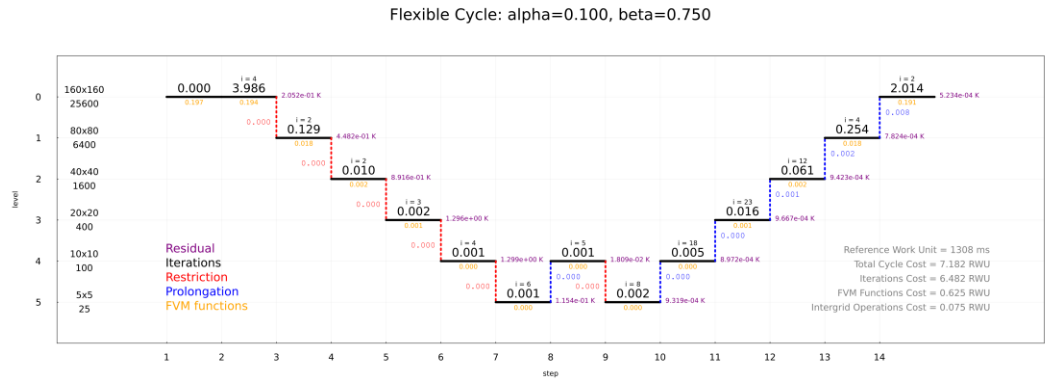


Run Summary Plot of Flexible Cycle → Alpha=0.05 Beta=0.85 → 8.29 RWU

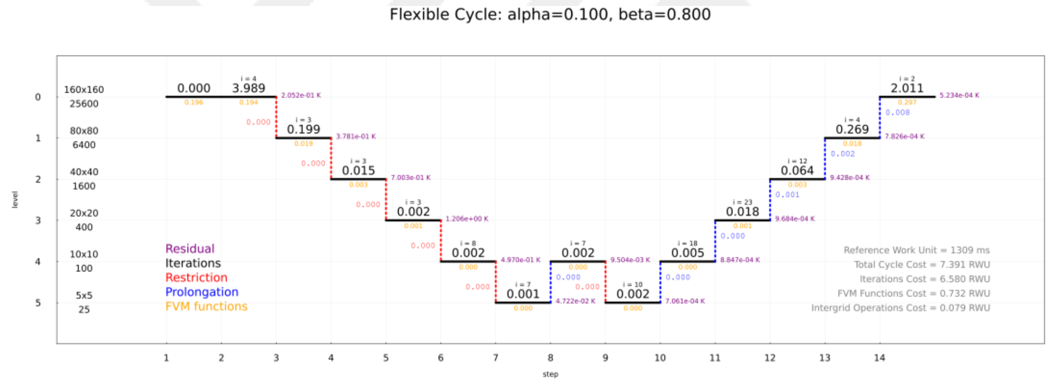


Appendix 6.2

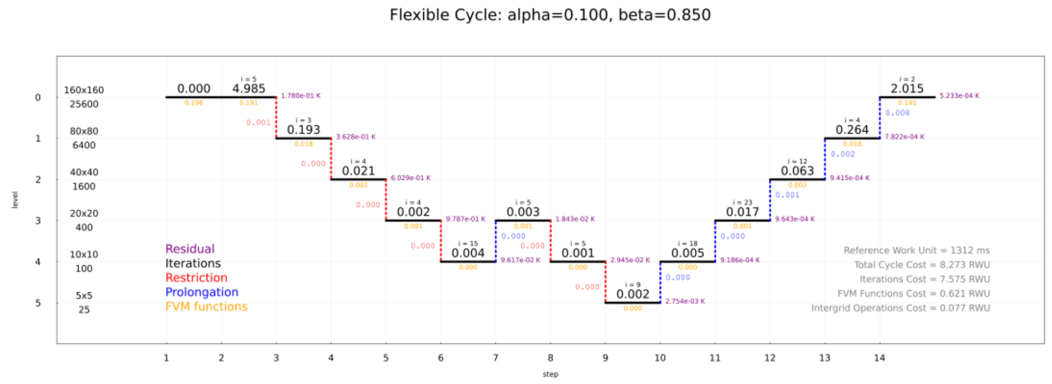
Run Summary Plot of Flexible Cycle → Alpha=0.10 Beta=0.75 → 7.18 RWU



Run Summary Plot of Flexible Cycle → Alpha=0.10 Beta=0.80 → 7.39 RWU

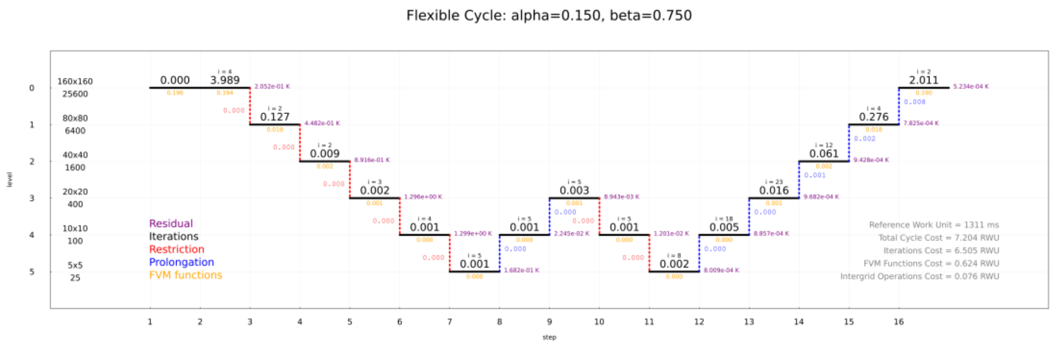


Run Summary Plot of Flexible Cycle → Alpha=0.10 Beta=0.85 → 8.27 RWU

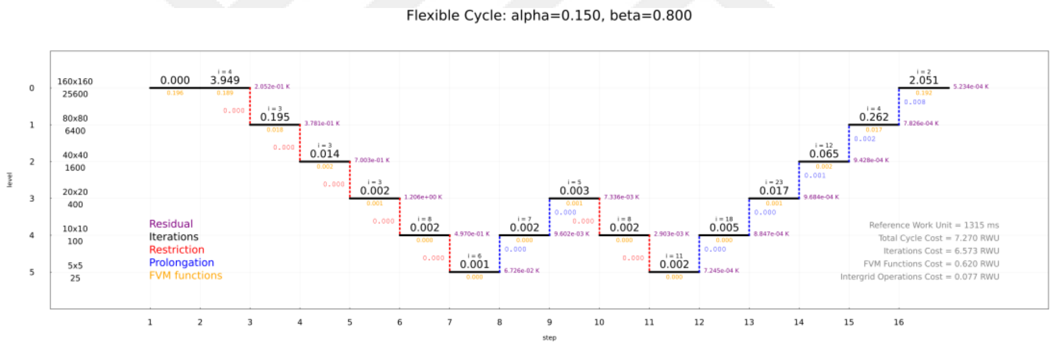


Appendix 6.3

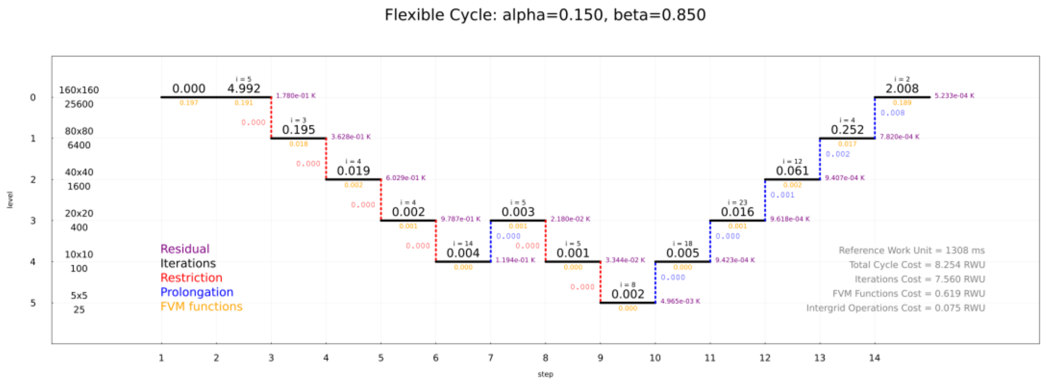
Run Summary Plot of Flexible Cycle → Alpha=0.15 Beta=0.75 → 7.20 RWU



Run Summary Plot of Flexible Cycle → Alpha=0.15 Beta=0.80 → 7.27 RWU

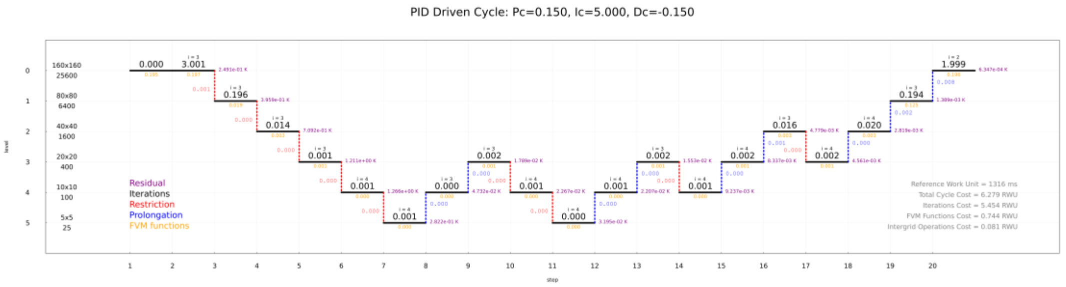


Run Summary Plot of Flexible Cycle → Alpha=0.15 Beta=0.85 → 8.25 RWU

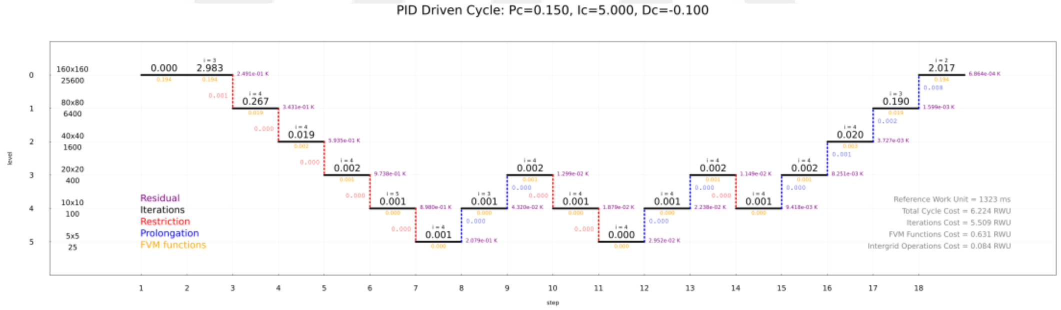


Appendix 7.1

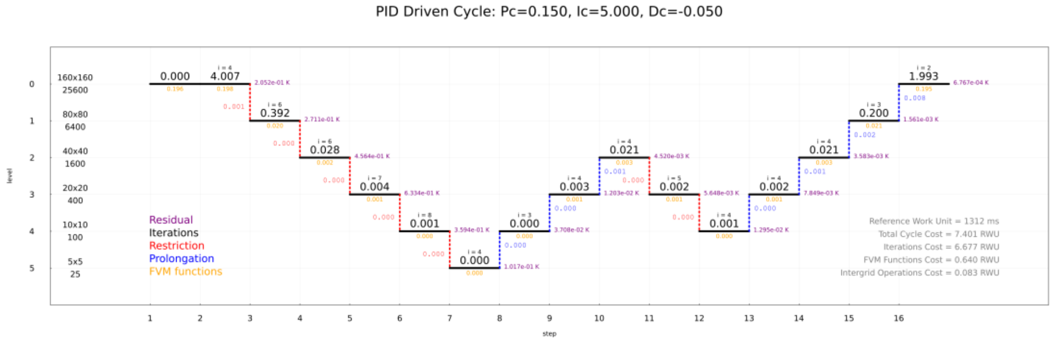
Run Summary Plot of PID Driven Cycle → Pc=0.15, Ic=5.0, Dc=-0.15 → 6.28 RWU



Run Summary Plot of PID Driven Cycle → Pc=0.15, Ic=5.0, Dc=-0.10 → 6.22 RWU

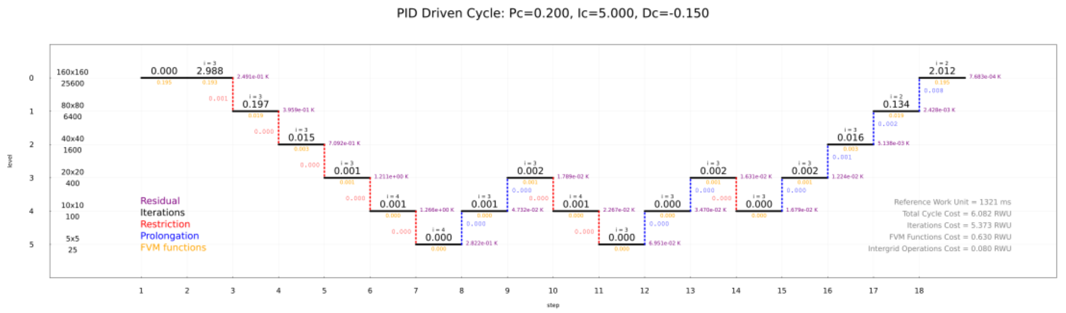


Run Summary Plot of PID Driven Cycle → Pc=0.15, Ic=5.0, Dc=-0.05 → 7.40 RWU

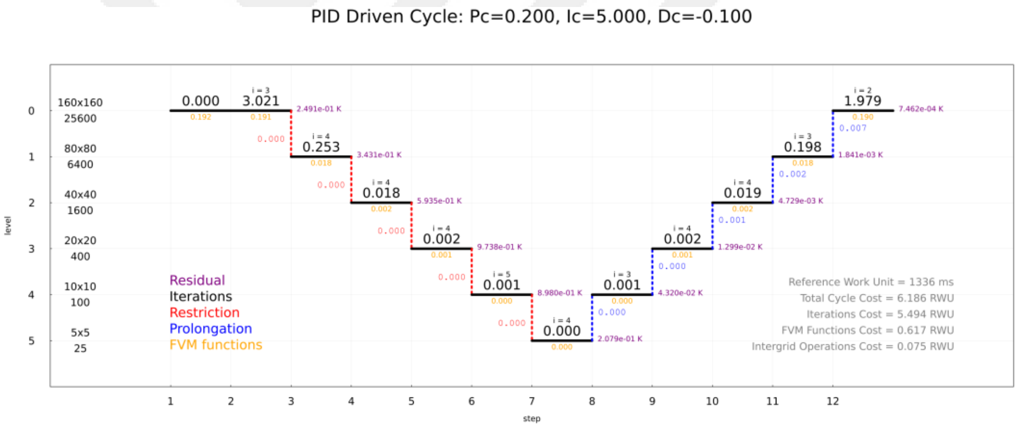


Appendix 7.2

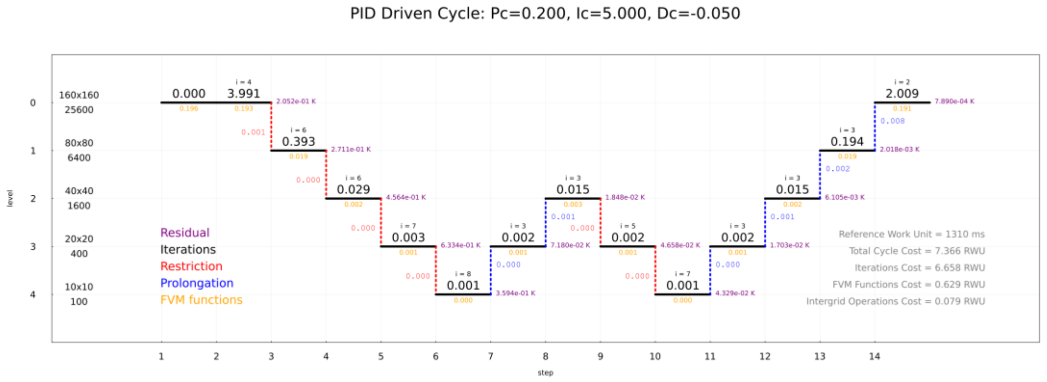
Run Summary Plot of PID Driven Cycle → Pc=0.20, Ic=5.0, Dc=-0.15 → 6.08 RWU



Run Summary Plot of PID Driven Cycle → Pc=0.20, Ic=5.0, Dc=-0.10 → 6.18 RWU

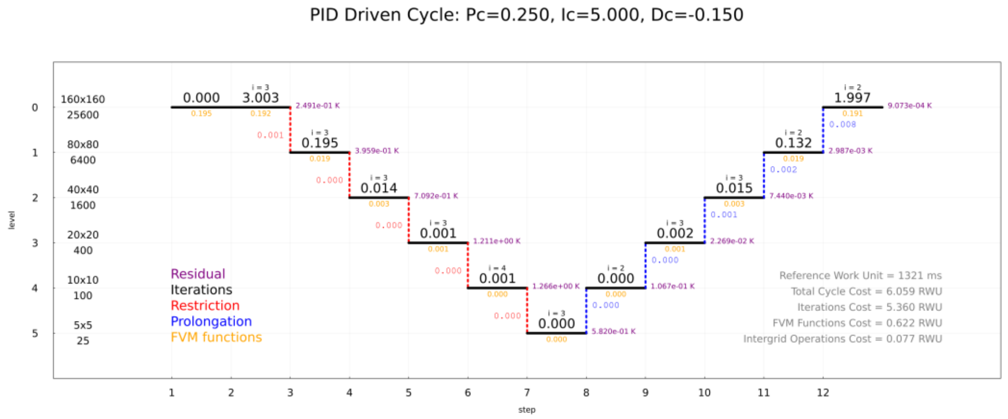


Run Summary Plot of PID Driven Cycle → Pc=0.20, Ic=5.0, Dc=-0.05 → 7.36 RWU

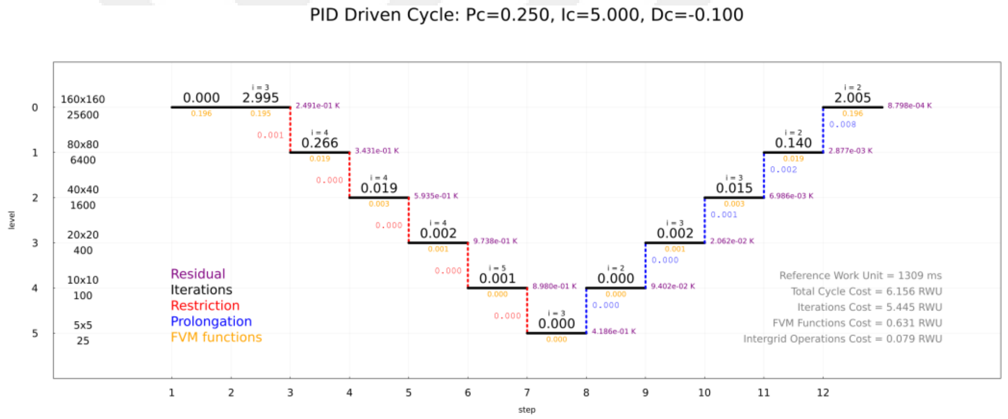


Appendix 7.3

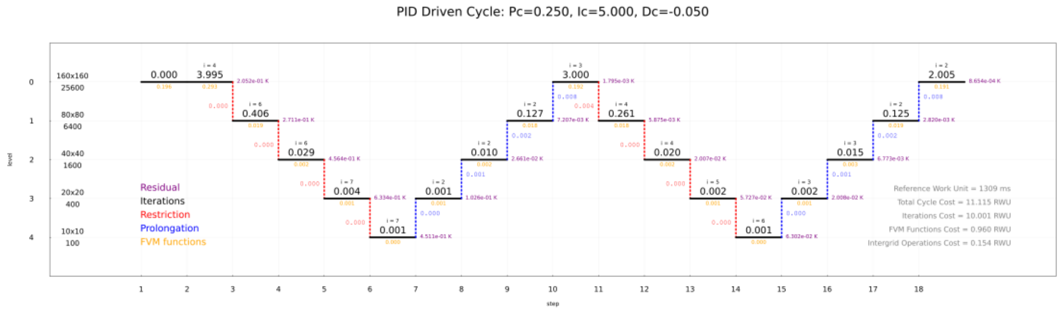
Run Summary Plot of PID Driven Cycle → Pc=0.25, Ic=5.0, Dc=-0.15 → 6.28 RWU



Run Summary Plot of PID Driven Cycle → Pc=0.25, Ic=5.0, Dc=-0.10 → 6.22 RWU

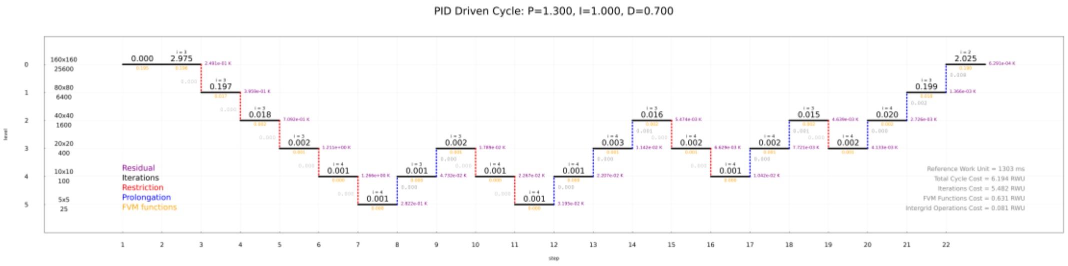


Run Summary Plot of PID Driven Cycle → Pc=0.25, Ic=5.0, Dc=-0.05 → 7.40 RWU

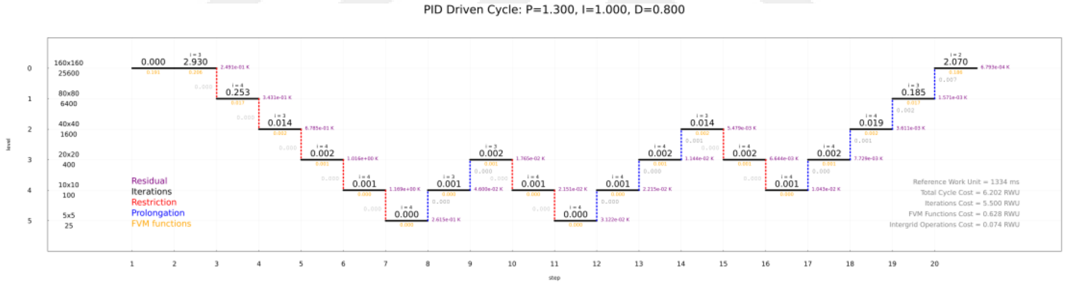


Appendix 8.1

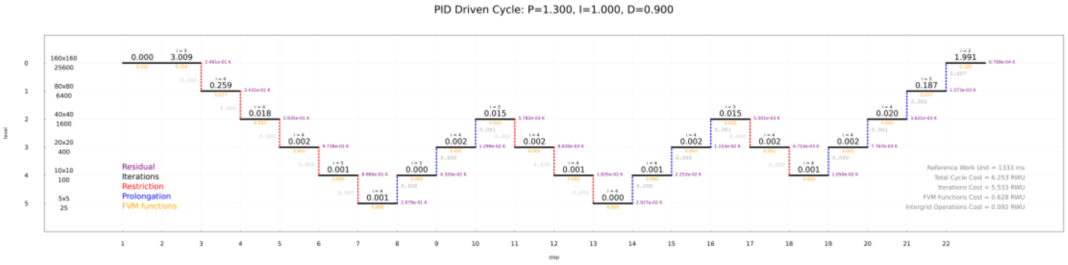
Run Summary Plot of PID Driven Cycle → cP=1.3, cI=1.0, cD=0.7 → 6.19 RWU



Run Summary Plot of PID Driven Cycle → cP=1.3, cI=1.0, cD=0.8 → 6.20 RWU

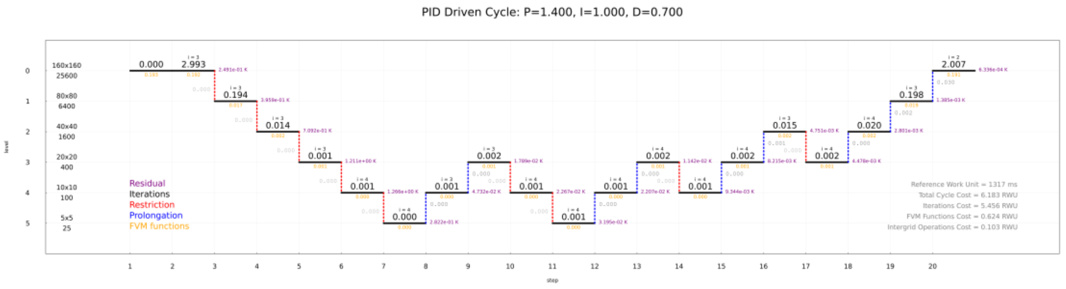


Run Summary Plot of PID Driven Cycle → cP=1.3, cI=1.0, cD=0.9 → 6.25 RWU

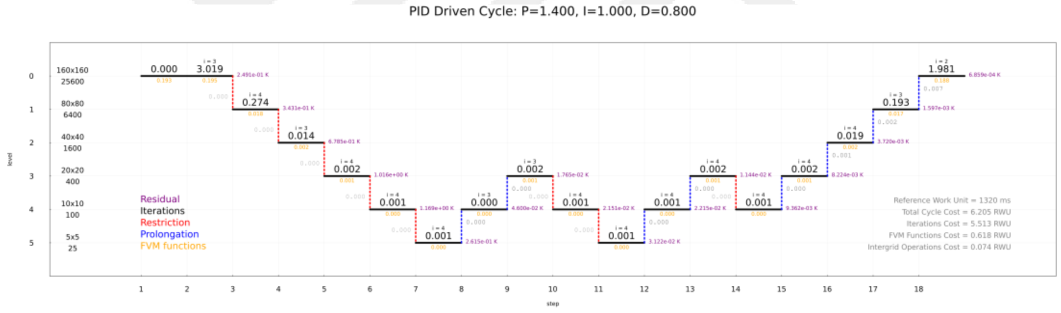


Appendix 8.2

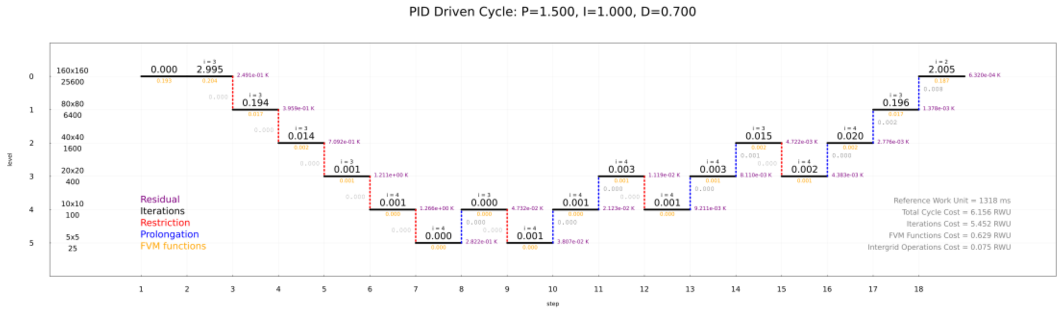
Run Summary Plot of PID Driven Cycle → cP=1.4, cI=1.0, cD=0.7 → 6.18 RWU



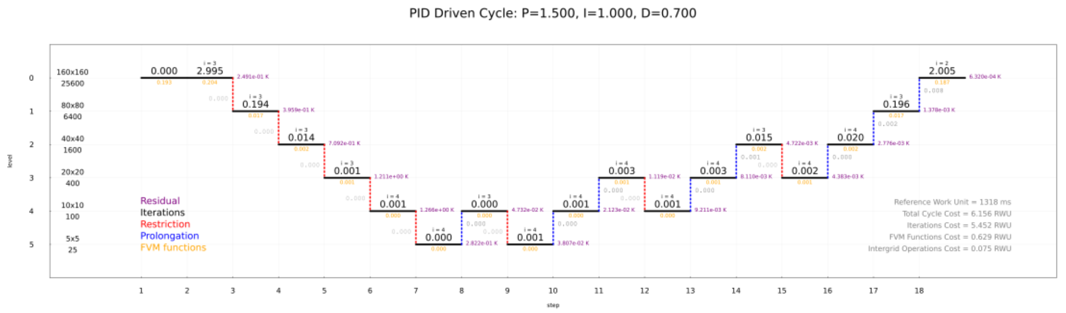
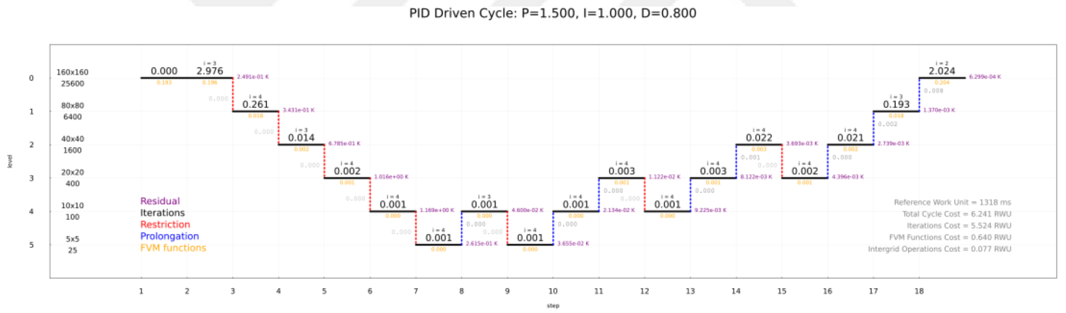
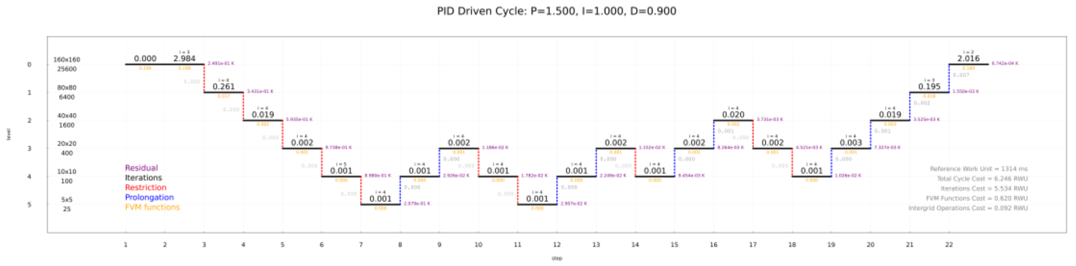
Run Summary Plot of PID Driven Cycle → cP=1.4, cI=1.0, cD=0.8 → 6.20 RWU



Run Summary Plot of PID Driven Cycle → cP=1.4, cI=1.0, cD=0.9 → 6.19 RWU



Appendix 8.3

Run Summary Plot of PID Driven Cycle → cP=1.5, cI=1.0, cD=0.7 → 6.15 RWU

Run Summary Plot of PID Driven Cycle → cP=1.5, cI=1.0, cD=0.8 → 6.24 RWU

Run Summary Plot of PID Driven Cycle → cP=1.5, cI=1.0, cD=0.9 → 6.25 RWU


CURRICULUM VITAE

ORCID ID: 0009-0002-2070-9036

Name Surname **: Mustafa Serdar TEKÇE**

Foreign Language **: English and Turkish (native)**

Professional Background:

- 2023 – 2024 (cont.), Lead Engineer, Turkish Aerospace Industries, Aircraft Division, Air Vehicle Engineering, Aerodynamics
- 2022 – 2023, Chief Engineer, Turkish Aerospace Industries, TF Division, Flight Sciences, Configuration Design and Development
- 2018 – 2022, Senior Engineer, Turkish Aerospace Industries, TF Division, Flight Sciences, Configuration Design and Development
- 2011 – 2018, Aeronautical Engineer, Anadolu University, Civil Aviation Research and Application Center

Educational Background:

- 2015 – 2024, Doctorate of Science, Eskişehir Technical University, Institute of Graduate Programs
- 2011 – 2015, Master of Science, Anadolu University, Graduate School of Science
- 2003 – 2010, Bachelor of Science, Istanbul Technical University, Aeronautical Engineering

Publications

- Tekçe M.S., Altuntaş Ö., Karakoç T.H., 2014, "*Investigation of Components' Aerodynamic Interaction of Aircraft via CFD for a small scale UAV*", WCUSEng.
- Şimsek D., Cerrah A.O., Ertan H., Tekçe M.S., 2013, "*The Assessment Of Postural Control Mechanisms In Three Archery Disciplines: A Preliminary Study*", PJSS.
- Tekçe M.S., Erdem D., 2010, "*Computational Investigation of Arrow Flight Aerodynamics", III. UHUK.*