ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PARAMETER OPTIMIZATION FOR MATHEMATICAL MODELING

Ph.D. THESIS

Mehmet TUNÇEL

Department of Mathematical Engineering

Mathematical Engineering Programme

JUNE 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PARAMETER OPTIMIZATION FOR MATHEMATICAL MODELING

Ph.D. THESIS

Mehmet TUNÇEL
(509132057)

Department of Mathematical Engineering

Mathematical Engineering Programme

Thesis Advisor: Prof. Dr. Ahmet DURAN

JUNE 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MATEMATİKSEL MODELLEME İÇİN PARAMETRE OPTİMİZASYONU

DOKTORA TEZİ

Mehmet TUNÇEL
(509132057)

Matematik Mühendisliği Anabilim Dalı

Matematik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Ahmet DURAN

HAZİRAN 2023

Mehmet TUNÇEL, a Ph.D. student of ITU Graduate School student ID 509132057 successfully defended the thesis entitled "PARAMETER OPTIMIZATION FOR MATHEMATICAL MODELING", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Ahmet DURAN** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. İsmail KÖMBE** Istanbul Commerce University	.....
	Asst. Prof. Dr. Bahri GÜLDOĞAN Istanbul Technical University	.....
	Prof. Dr. Can ÖZTURAN Bogazici University	.....
	Prof. Dr. Burak SALTOĞLU Bogazici University	.....

Date of Submission : **24 May 2023**
Date of Defense : **9 June 2023**

To the memory of my father,

FOREWORD

This thesis is lovingly dedicated to my wife Havva TUNÇEL, to my parents Gündoğar and Veli TUNÇEL, and to my sons Ömer and Muhammed. I would like to express my sincere gratitude to them in particular for their compassion, and encouragement.

I would like to express my gratitude to Prof. Dr. Ahmet DURAN, my respected adviser, for his support, inspiration, tolerance, and direction throughout my Ph.D. studies. He is one of the pioneers of Quantitative Behavioral Finance discipline using differential equations, statistical methods, mathematical models and parameter optimization. During my higher education, I would like to express my sincere thanks once again to my esteemed advisor Prof. Dr. Ahmet DURAN for giving me the opportunity to work on the following international projects as a researcher:

- Parallel Algorithm (Kernel) Development for Large Scale Sparse Linear Systems in Oil Reservoir Simulation (PASSOR), Computational Linear Algebra Project for ARAMCO Overseas, 2012-2017.
- HPC based Design of a Novel Electromagnetic Stirrer for Steel Casting, EU Project, PRACE-4IP , 2015 - 2016.
- Scalable Parallel Nonlinear Parameter Optimization Algorithm with Parameter Pools, EU Project, PRACE-2IP WP12 T12.2, 2013 - 2014.
- Scalability of OpenFOAM for Bio medical Flow Simulations, EU Project, PRACE-3IP WP7 T7.2, 2012 - 2014.
- SuperLU MCDT Many Core Distributed Solver on MIC Architecture, EU Project, PRACE-1IP WP7 T7.1, 2013 - 2013.
- Structural Analysis of Large Sparse Matrices for Scalable Direct Solvers, EU Project, PRACE-2IP WP12 T12.2, 2011 - 2013.
- Performance Analysis of BLAS Libraries in SuperLU DIST for SuperLU MCDT Multi Core Distributed Development, EU Project, PRACE-2IP WP12 T12.2, 2011 - 2013.
- Scalable and Improved SuperLU on GPU for Heterogeneous Systems, EU Project, PRACE-2IP WP12 T12.2, 2011 - 2013.
- Design and Implementation of New Hybrid Algorithm and Solver on CPU for Large Sparse Linear Systems, EU Project, PRACE-2IP WP12 T12.2, 2011 - 2013

I really thank the Doctoral Committee members Professor Dr. İsmail KÖMBE from Istanbul Commerce University, Asst. Prof. Dr. Bahri GÜLDOĞAN from Istanbul Technical University, Prof. Dr. Can ÖZTURAN from Bogazici University, Prof. Dr.

Burak SALTOĞLU from Bogazici University and especially Prof. Dr. Ahmet DURAN for providing helpful comments and suggestions. And I would like to thank Prof. Dr. Fatma ÖZDEMİR from Istanbul Technical University, Prof. Dr. Mine ÇAĞLAR from Koc University, Prof. Dr. Mustafa S. ÇELEBİ from Istanbul Technical University and Asst. Prof. Dr. Şenol PİŞKİN from Istinye University.

Sincere appreciation to my friends and colleagues for their helpful communication.

I would also like to thank Saudi Aramco for sponsoring a part of the research and the Saudi Aramco visit. I'm grateful to Dr. Ali DOGRU for the project coordination, valuable comments, and suggestions. I thank Larry FUNG for helpful communications, constructive feedback, and for providing the reservoir simulation test matrices.

I'm thankful to Istanbul Technical University (ITU) - the National Center for High-Performance Computing of Turkey (UHeM) for computing resources used partly in the study of Chapter 3 and 5.

This research was partly supported by the PRACE-2IP project funded in part by the EU's 7th Framework Programme (FP7/2011-2014) under grant agreement no. RI-283493 and Project 2010PA2507 were awarded under the 19th Call for PRACE Preparatory Access. The suggestions of the editors and two anonymous referees are also appreciated.

Also, this research was supported as a Ph.D. thesis project under project ID 1323 and project code 39332 in part by Istanbul Technical University - Scientific Research Project (ITU – BAP). It has been gratefully acknowledged throughout my Ph.D. education.

June 2023

Mehmet TUNÇEL
(Lecturer)

TABLE OF CONTENTS

ABBREVIATIONS

AFDE : Asset flow differential equation
BFGS : Broyden-Fletcher-Goldfarb-Shanno
CEF : Closed-end fund
CPU : Central processing unit
CUDA : Compute unified device architecture
EOS : Equation of state
FLOPS : Floating point operations per second
GPU : Graphics processing unit
IPV : Initial parameter vector
IVP : Initial value problem
MP : Market price
MIF : Maximum improvement factor
MPI : Message passing interface
NAV : Net asset value
NLS : Nonlinear least squares
NYSE : New York Stock Exchange
OpenMP : Open multi-processing
PCA : Principal component analysis
QN : Quasi-Newton
RK4 : Runge-Kutta fourth order

SYMBOLS

c_1	: Time scale coefficient for the momentum
c_2	: Time scale coefficient for the valuation
c_{iGlOpt}	: Minimum error of the NLS error function F for the i^{th} event
K	: Parameter vector
K_{iGlOpt}	: Optimal parameter vector for c_{iGlOpt}
$\mathbb{K}$	: Parameter vector pool
$\mathbb{N}$	: Natural numbers
q_1	: Coefficient of the trend-based sentiment
q_2	: Coefficient of the value-based sentiment
$\mathbb{R}$	: Real numbers
h_{RK4}	: RK4 step size
ε_1	: Threshold for the gradient
ε_2	: Threshold for the nonlinear least squares error

LIST OF TABLES

LIST OF FIGURES

PARAMETER OPTIMIZATION FOR MATHEMATICAL MODELING

SUMMARY

Mathematical modeling is used to explain and forecast complex systems, and parameter optimization methods have a crucial role to find the optimal set of parameters obtained by minimizing an objective function. Also, the management of computational resources is essential for handling big models in real-time scenarios.

A. Duran and G. Caginalp (2008) propose a hybrid parameter optimization forecast algorithm for asset prices via asset flow differential equations. In this thesis, we propose a new mathematical method for an inverse problem of parameter vector optimization in asset flow theory. For this purpose, we use quasi-Newton (QN) and Monte Carlo simulations to optimize the function $F[\tilde{K}]$ for each selected event and initial parameter vector. We present grid and random methods and conclude that the grid approach is better than the random approach in the unconstrained optimization problem.

This study also presents a parallel numerical parameter optimization algorithm for dynamical systems used in financial applications. It achieves speed-up for up to 512 cores and considers more extensive financial market situations. Moreover, it also evaluates the convergence of the model parameter vector via nonlinear least squares error, and maximum improvement factor.

In this thesis, we also examine the performance, scalability, and robustness of OpenFOAM on the GPGPU cluster for bio-medical fluid flow simulations. It compared the CPU performance of iterative solver icoFoam with direct solver SuperLU_DIST 4.0 and hybrid parallel codes of MPI+OpenMP+CUDA versus MPI+OpenMP implementation of SuperLU_DIST 4.0. Results showed speed-up for large matrices up to 20 million x 20 million.

Besides that, we investigate the usage of eigenvalues to examine the spectral effects of large matrices on the performance of scalable direct solvers. Gerschgorin's theorem can be used to bound the spectrum of square matrices, and behaviors such as disjoint, overlapped, or clustered Gerschgorin circles can give clues. We define the minimum number of cores and show that it depends on the sparsity level and size of the matrix, increasing slightly as the sparsity level decreases and the order increases.

In sum, this thesis presents new methods for initial parameter selection and a new algorithm for parallel numerical parameter optimization. Also, we define new metrics and show that the importance of right matching for computational systems and the optimal minimum number of cores are important in mathematical modeling and simulation.

MATEMATİKSEL MODELLEME İÇİN PARAMETRE OPTİMİZASYONU

ÖZET

Dinamik sistemleri, diferansiyel denklemleri ve istatistiksel modelleri içinde barındırabilen matematiksel modellemeler fen bilimleri, mühendislik ve finansın birçok alanında karşımıza çıkan karmaşık sistemlerin davranışını yorumlamak ve tahmin etmek için etkili yöntemlerdendir. Model parametrelerinin tahmini matematiksel modellemenin kritik parçasıdır. Bu parametreler model doğruluğu ve performansı üzerinde önemli etkiye sahiptirler. Belirli bir objektif fonksiyonu için en optimal değerleri sağlayan parametreleri belirlemek, parametre optimizasyon yöntemlerinin amacıdır ve sınırlı bir sürede anlamlı parametrelerle daha fazla doğrulukta sonuca yakınsamak birçok gerçek hayat uygulaması için önemsenen stratejik amaçlardandır. Bunun yanı sıra, büyük doğrusal sistem denklemlerinin çözümü, bazı matematiksel modeller için önemli adımlardandır. Bu nedenle, hesaplama kaynakların yönetimi, matematiksel modelleme için de önemli bir aşama olarak karşımıza çıkmaktadır.

Etkin başlangıç parametre vektörlerinin seçimi, birçok bilim ve mühendislik probleminde parametre vektörlerine ve diferansiyel denklemlere sahip matematiksel modeller için önemli yere sahiptir. A. Duran ve G. Caginalp (2008) varlık akış diferansiyel denklemleri ile hisse fiyatı için hibrit parametre optimizasyon tahmin algoritması sundu. Bölüm 2'de, parametre vektör optimizasyonunun ters bir problemi için yeni bir matematiksel yöntem öneriyoruz. Hiper kutudaki ızgara ve rasgele yaklaşımların etkinliğini, varlık akışı teorisinden gelen matematiksel bir modelde parametre vektör optimizasyonunun ters bir problemi için doğrusal olmayan en küçük kareler hatası, maksimum iyileştirme faktörü ve yineleme sayısı açısından analiz ediyor ve karşılaştırıyoruz. Bu analiz, yatırımcılar ve makine öğrenimi uygulamalarında popülasyon dinamiklerinin anlaşılması açısından oldukça değerlidir. Bu amaçla, seçilen her olay ve başlangıç parametre vektörü için $F[\tilde{K}]$ fonksiyonunu optimize ediyoruz. Burada geri izleme satırı arama algoritmasını kullanan Broyden-Fletcher-Goldfarb-Shanno (BFGS) formülüne sahip quasi-Newton (QN) yöntemini kullanıyoruz. $F[\tilde{K}]$ simülasyon yoluyla hesaplanıp gerçek piyasa fiyatını temsil eden değerleri ile hesaplanan piyasa fiyatı değerleri arasındaki üstel ağırlıklı kare farkların toplamını temsil etmektedir. Bu çalışmamızda (A. Duran ve G. Caginalp, 2008)'den farklı olarak, Monte Carlo simülasyonları ve yakınsama diyagramları elde ettik. Bunları kullanarak ve sınırsız optimizasyon problemindeki simülasyon veri setimize dayanarak ızgara yaklaşımının başarısının rastgele yaklaşıma nispeten daha iyi olduğunu görmekteyiz.

Bunun yanı sıra, kısa zamanda karar vermenin çok önemli olduğu finansal uygulamalarda kullanılan dinamik bir sistem için ölçeklenebilir bir paralel sayısal parametre optimizasyon algoritmasına sahip olmak önemlidir. Bölüm 3'te, Message

Passing Interface (MPI) paralel programlamasını kullanıyoruz ve parametre tahmini için böyle yeni bir paralel algoritma sunmaktayız. Algoritmamızı 1989'dan beri G. Caginalp ve araştırma ekibi tarafından geliştirilen ve analiz edilen varlık akışı diferansiyel denklemlerine uygulamaktayız. Bazı zaman serileri için (A. Duran and M. Tunçel, 2014)'de 512 çekirdeğe kadar hızlanma sağlanmakla birlikte (A. Duran and M. Tunçel, 2014)'den farklı olarak, bu çalışmada daha kapsamlı finansal piyasa durumlarını, örneğin düşük volatilite, yüksek volatilite ve borsa fiyatının değişen büyüklüklerine göre net varlık değerinde iskonto/prim durumlarını da ele alıyoruz. Ayrıca, ilk parametre vektörlerinin sayısına bağlı olarak optimizasyon işleminin başarısını ölçmek için model parametre vektörünün yakınsamasını, doğrusal olmayan en küçük kareler hatasını ve maksimum iyileştirme faktörünü de değerlendirmekteyiz.

Büyük matematiksel modeller için ele alınması gereken bir konu da hesaplama sistemlerinin doğru eşleştirilmesi ve parametrelerinin ayarlanmasıdır. Bu konu için Bölüm 4'te farklı yüksek performanslı hesaplama kümelerinde arterlerdeki kan akışının simülasyonundan gelen büyük matrislerin çözümü için biyomedikal sıvı akışı simülasyonlarının ve OpenFOAM'da çözücü olan icoFoam'un hesaplama zorluklarını incelemekteyiz. Akış problemi simülasyonunda üretilen matrisler zaman ilerledikçe her adımda birbirinden farklı matematiksel özelliklere geçiş yapmaktadır. Bu çalışmada, çözücülerin kötü koşullu (ing: ill-conditioned) matrisler için davranışlarını inceledik. Yinelemeli çözücü icoFoam'un ve doğrudan çözücü olan SuperLU_DIST'in hibrit paralel kodlarını (MPI + OpenMP) CPU performanlarını Fransa CEA TGCC'nin Curie (Tier-0 sistemi) ince düğümlerinde koşturarak karşılaştırdık. Ayrıca, TGCC'nin Curie (Tier-0 sistemi) hibrit düğümlerinde SuperLU_DIST'in MPI + OpenMP ve MPI + OpenMP + CUDA hibrit paralel çözücü kodlarının performansını farklı parametreler ile inceleyip karşılaştırdık. 20 milyon x 20 milyona kadar olan büyük matrisler için çözücülerin hızlandırılmasına ilişkin sonuçları bu bölümde irdelemekteyiz.

Matematiksel modellemelerin içerdiği diğer bir önemli konu da petrol ve gaz rezervuarı simülasyonlarında olduğu gibi zaman kısıtlı gerçek hayat karar verme uygulamaları için büyük seyrek doğrusal sistemleri tahmin edilebilir bir sürede performanslı bir şekilde çözmek için gerekli ön bilgileri ve parametreleri elde etmektir. Bu nedenle, büyük matrislerin ölçeklenebilir doğrudan çözücülerin performansı üzerindeki spektral etkilerini özdeğerleri kullanarak incelemeyi amaçladık. Bölüm 5'te, bir matrisin özdeğer dağılımı ile çözücünün performansı arasında bir ilişki olup olmadığını araştırmaktayız. Çeşitli seyrek matrislerin özdeğer dağılımlarını ele aldık. Bazen özdeğerlerin dağılım grafiğini elde etmek için tüm özdeğerler bulunabilmektedir. Ama büyük matrisler için tüm özdeğerleri bulmak hesaplama ve kaynakları açısından oldukça pahalıdır. Bu nedenle, Gerschgorin teoremi kare matrislerin spektral durumunun tahmini için kullanılabilmektedir. Gerschgorin dairelerinin ayrık, üst üste binmesi veya kümelenmesi gibi çeşitli davranışlar, özdeğerlerin dağılımı ve çözücünün bu matris için performansı hakkında ipucu verebilmektedir. Bu merkezde Bölüm 5'te, rastgele doldurulmuş seyrek matrisleri ve rezervuar modellemesinden gelen çeşitli desenli matrisleri içeren bir test matrisleri portföyünü tek gözeneklilik tek geçirgenlikden çift gözeneklilik çift geçirgenlik modellerine kadar ele almaktayız. 3 fazlı modelden ve 7 bileşenli EOS modelinden desenli matrislere ek olarak Florida Üniversitesi seyrek matris koleksiyonundan modifiye edilmiş HELM2D03LOWER_20K matrisimizi ve EMILIA_923 matrislerini

ayrıntılı incelemekteyiz. En uygun minimum çekirdek sayısını, belirli bir problem boyutu için minimum çözüm süresini sağlayan çekirdek sayısı olarak tanımlamaktayız. Burada problem boyutu ile matrisin spektral etkileri ve bellek gibi kullanılabilir kaynaklar arasında bir ilişki görünmektedir. Gerekli en uygun minimum çekirdek sayısının, matrisin seyreklik seviyesine ve boyutuna bağlı olduğunu görüyoruz. Matrisin seyreklik seviyesi azaldıkça ve matrisin boyutu arttıkça, optimal minimum çekirdek sayısının artmasını beklemekteyiz.

Sonuç olarak yukarıdaki çalışmaları içeren bu tez, parametre optimizasyon yöntemleri ve matematiksel modellemede yaygın olarak kullanılan doğrusal denklem çözümleri için önemli olan yeni yöntem ve stratejiler sunmaktadır. Her bölüm, çalışmanın ayrıntılarıyla ilgili ayrı ayrı literatür taramasını da içermektedir. Tez özetle aşağıdaki gibi düzenlenmiştir. Bölüm 2'de, daha iyi yakınsama performansı elde etmek için parametre optimizasyonunda kullanılan başlangıç parametre vektörü seçimi için yeni bir yöntem sunmaktayız. Bölüm 3'te, matematiksel modellemenin çözümü için çok önemli olan zaman sınırlamasını dikkate alarak paralel sayısal parametre optimizasyonu için yeni bir algoritma önermekteyiz. Bölüm 4'te, hesaplamalı sistem için doğru kaynakların ve kombinasyonlarının eşleştirilmesinin önemi, yaygın olarak kullanılan senaryolar ve testlerle gösterilmektedir. Bölüm 5'te, hesaplamalı kaynak büyüklüğü ile problemin doğru ve verimli eşleşmesi, özdeğer spektrumu kullanılarak doğrusal sistemin ön değerlendirmesi ile incelenmiş ve optimal minimum çekirdek sayısı tanımlanmıştır. Bölüm 6'da bu tezden elde edilen sonuçlar bütünüyle ele alınıp değerlendirilmektedir.

1. INTRODUCTION

Mathematical modeling including dynamical systems, differential equations and statistical models is an effective method for explaining and forecasting the behavior of complex systems in many disciplines of science, engineering, and finance.

The estimation of model parameters, which can have a major impact on model correctness and performance, is a critical component of mathematical modeling [1–3]. Finding the optimal set of parameter values to minimize a specified objective function is the goal of parameter optimization methods, and more accuracy with meaningful parameters in a limited time is the strategy for many real-life applications [4]. Besides that, the solution for large system of linear equations is an important operation for fitting the mathematical model with real use cases. Hence, the management of computational resources arises as an important operational case for mathematical modeling.

The selection of effective initial parameter vectors is important for mathematical models having parameter vectors and differential equations in many science and engineering problems. In Chapter 2, we propose a new mathematical method for an inverse problem of parameter vector optimization [5]. We analyse and compare the effectiveness of grid and random approaches in hyperbox in terms of nonlinear least squares error, maximum improvement factor and number of iterations for an inverse problem of parameter vector optimization in a mathematical model coming from asset flow theory. This analysis is valuable to understand the population dynamics of investors and machine learning applications. For this purpose, we use quasi-Newton (QN) method having the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula with backtracking line search algorithm to optimize the function $F[\tilde{K}]$ for each selected event and initial parameter vector, where $F[\tilde{K}]$ represents the sum of exponentially weighted squared differences between the proxy for actual market price values via simulation and the computed market price values. Moreover, we employ Monte Carlo

simulations and obtain convergence diagrams. We find that the success of the grid approach is relatively better than that of the random approach based on our simulation data set in the unconstrained optimization problem.

It is important to have a scalable parallel numerical parameter optimization algorithm for a dynamical system used in financial applications where time limitation is crucial. The asset flow differential equations that have been developed and analyzed since 1989 (see [4,6–9] and references contained therein). The asset flow differential equations have several versions. In Chapter 3, we use Message Passing Interface parallel programming and present such a new parallel algorithm for model parameter estimation [10]. For example, we apply the algorithm to the 3rd version of the asset flow differential equations (see [7,8]). We achieve speed-up for some time series to run up to 512 cores (see [11]). Unlike [11], we consider more extensive financial market situations, for example, in presence of low volatility, high volatility and stock market price at a discount/premium to its net asset value with varying magnitude, in this work. Moreover, we examine the convergence of the model parameter vector, the nonlinear least squares error and maximum improvement factor to quantify the success of the optimization process depending on the number of initial parameter vectors.

The right matching of computational systems and tuning their parameters are important to handle large mathematical models. So, we cope with the computational challenges for bio-medical fluid flow simulations and an OpenFOAM solver, icoFoam, for the large matrices coming from the simulation of blood flow in arteries on different HPC clusters in Chapter 4 [12]. The flow problem produces different kind of matrices as the time advances in extensive simulation. In this study we examine the behaviour of the solvers for ill-conditioned matrices. We compare the CPU performance of the iterative solver icoFoam and the hybrid parallel codes (MPI+OpenMP) of a direct solver SuperLU_DIST 4.0 (see [13]) at TGCC Curie (a Tier-0 system) thin nodes at CEA, France (see [14]). Moreover, we compare the performance of the hybrid parallel codes of MPI+OpenMP+CUDA versus MPI+OpenMP implementation of SuperLU_DIST 4.0 at TGCC Curie (a Tier-0 system) hybrid nodes of CPU + GPU at CEA, France (see [14]). We discuss the performance, scalability and robustness of OpenFOAM on

GPGPU cluster. We show our results about the speed-up of the solvers for the large matrices of size up to 20 million x 20 million.

It is valuable to estimate the elapsed time to solve large sparse linear systems for time-restricted real life decision making applications such as oil and gas reservoir simulators. Challenging matrices should be distinguished and handled separately because they may lead to performance bottleneck. Therefore, we need to examine the spectral effects of large matrices on the performance of scalable direct solvers by using eigenvalues. In Chapter 5, we check whether there is relationship between the eigenvalue distribution of a matrix and the performance of the solver [15]. We try to examine the eigenvalue distribution of various sparse matrices. We may find all eigenvalues in order to obtain the distribution graph of eigenvalues, if possible. However, it is very expensive to find all eigenvalues. Therefore, Gerschgorin's theorem may be used to bound the spectrum of square matrices. Several behaviors such as being disjoint, overlapped or clustered of Gerschgorin circles may give clue regarding the distribution of the eigenvalues and the performance of the solver for that matrix.

In Chapter 5, we consider a portfolio of test matrices which include randomly populated sparse matrices and various patterned matrices coming from reservoir modeling from single porosity single permeability to dual porosity dual permeability models (see [16]). We examined our modified HELM2D03LOWER_20K matrix and EMILIA_923 matrix from the University of Florida sparse matrix collection (see [17]), in addition to the patterned matrices from 3 phase black-oil model and 7 component EOS model. We define an optimal minimum number of cores as the number of cores that provides the minimum wall clock time for a given size of problem, where a right match occurs between the problem size, the spectral effects of matrix and the available resources such as memory, in presence of communication overhead. We find that the optimal minimum number of cores required depends on the sparsity level and size of the matrix. As the sparsity level of matrix decreases and the order of matrix increases, we expect that the optimal minimum number of cores increases slightly.

In these contexts, this thesis provides new methods and strategies which are crucial for parameter optimization methods and linear equation solutions commonly used in

mathematical modeling. Each chapter also contains its literature review separately related to the details of the study.

The remainder of the thesis is organized as follows. In Chapter 2, we present a new method for initial parameter selection used for parameter optimization to get better convergence performance. In Chapter 3, we propose a new algorithm for parallel numerical parameter optimization to handle time limitation that is crucial for the solution of the mathematical modeling. In Chapter 4, the importance of right matching for the computational system is demonstrated with commonly used scenarios and tests. In Chapter 5, right matching for the computational resource is examined with pre-evaluation of the linear system using the eigenvalue spectrum, and the optimal minimum number of cores is defined. Chapter 6 concludes the thesis.

2. EFFECTIVENESS OF GRID AND RANDOM APPROACHES FOR A MODEL PARAMETER VECTOR OPTIMIZATION

It is important to understand the dependence on initial parameter vector values of a nonlinear dynamical system for an inverse problem of parameter vector estimation in science and engineering problems. In this chapter, we propose a new mathematical method and focus on Monte Carlo simulation to find out the effectiveness of two approaches including grid approach and random approach in hyperbox based on our experimental design for selection of initial parameter vectors in a large-scale unconstrained optimization problem. The study in this chapter was published in "Journal of Computational Science" with title "Effectiveness of grid and random approaches for a model parameter vector optimization" [5].

Numerical optimization methods in inverse problems and simulations play important role in many science, engineering and econophysics applications. For example, [2] proposes a method for estimation of biochemical kinetics parameters with treatment of initial value problem (IVP) simulation for a system of nonlinear ordinary differential equations. Moreover, parameter vector optimization is a central part of machine learning applications in econophysics, mathematical finance and economics (see [4,18–20]).

Duran and Caginalp [4] propose a hybrid parameter optimization forecast algorithm including daily based learning with two streaming windows such as long window of most recent days (for example, 10-day window) to compute the relative valuation change and short window (for example, 5-day window) to compute optimal parameter vector, using a semi-dynamic initial parameter vector pool $\mathbb{K}$ having not only fixed but also most recently used successful parameter vectors from a set of grid points in a hyper-box and out-of-sample prediction.

The coefficient, q_1, for the trend-based investors' sentiment is the dominant parameter for the market price according to the forward sensitivity analysis done by Duran [21].

Figure 2.1 : Gunduz Caginalp (1952 - 2021)

Such parameters like q_1, c_1, q_2 and c_2 should be obtained via suitable parameter optimization techniques in a mathematical model having differential equations.

In literature, there are various approaches such as multi-start methods (see [22,23] and hyperbox methods [24] for different global unconstrained optimization problems [25]. Considering global optimization problem on the multidimensional space, the selection of the initial parameter vectors has critical importance to converge a candidate solution in a reasonable time/iteration for real-time applications. At this point, another important issue is that the appropriate initial parameter vectors in a feasible region should be selected so that they can generate candidate feasible solutions that are especially meaningful for the real-world problems.

The parameter optimization algorithm solves the complex stiff problem of nonlinear dynamical system called as the asset flow differential equations (AFDEs) and optimize its parameters for a certain interval of day [4]. AFDEs have been developed by Caginalp (see Figure 2.1) and collaborators since 1989 [6,7]. This important mathematical model may describe different nonlinear behaviors of asset markets (see [26–28]). The dynamical microeconomic model suggests valuable constraints analogous to conservation laws in physics, instead of the classical time series analysis with a single stage approach (see [4]).

Duran [29] and Duran and Caginalp [4] introduced a serial algorithm called the asset flow optimization forecast algorithm. Later, Duran and Tuncel [10] proposed message passing interface (MPI) based scalable parallel algorithm for parameter optimization of AFDEs with fixed parameter pools using central processing unit (CPU).

In this study, we use time series of market price and net asset value data obtained via our Monte Carlo simulation rather than real closed-end fund data, since the simulation may capture more various scenarios than that of real data for a particular time interval, unlike Duran and Caginalp [4]. Moreover, we do not use learning algorithm. We have static initial parameter vector pools instead of semi-dynamic pools, in order to examine the impact of the static pools. Our goal is to explore information about optimal/feasible parameters in various market scenarios based on asset flow theory.

We use the first four moments including mean, standard deviation, skewness and kurtosis in addition to minimum and maximum values for MP and NAV as an input feature, unlike Duran and Caginalp [4]. Volatility is measured by standard deviation of the daily closing price time series. Moreover, as a binary feature we use the discount and premium status as 1 and 0, respectively. Furthermore, we apply the principal component analysis (PCA) in order to detect the pattern of the dominant features and check the representation success of the initial parameter vector pool. We project the high dimensional feature space into 2D and 3D spaces using PCA.

To the best of our knowledge, this is the first study to compare the grid and random approaches in hyperbox for parameter optimization of asset flow differential equations. We find that the grid approach is relatively better than the random approach in our data set. Our study is important to develop an optimization software for algorithmic trading.

The remainder of this chapter is organized as follows. In Section 2.1, parameter optimization algorithm for the dynamical system is introduced. In Section 2.2, we present the related theorems for the convergence of the nonlinear least squares technique for the differential equations. In Section 2.3, we illustrate our design of experiments and the principal component analysis. In Section 2.4, we apply the parameter optimization algorithm with two different approaches for initial parameter vector pools. We show the test results for the performance of the two approaches

Table 2.1 : Input variables and their descriptions for Algorithm 1.

Variable name	Description
$\mathbb{K}$	Parameter vector pool
MP	Sequence of market prices
NAV	Sequence of net asset value prices
i	Event number to be searched for optimal parameter
n	Short window size to compute optimal parameter vector
m	Long window size to compute relative valuation change
h_{RK4}	RK4 step size
ε_1	Threshold for the gradient
ε_2	Threshold for the nonlinear least squares error

Table 2.2 : Output variables and their descriptions for Algorithm 1.

Variable name	Description
K_{iGlOpt}	Optimal parameter vector for c_{iGlOpt}
c_{iGlOpt}	Minimum error of the NLS error function F for the i^{th} event
MIF_{iGlOpt}	MIF for optimal parameter vector K_{iGlOpt}
$QNiter_{iGlOpt}$	QN iteration for optimal parameter vector K_{iGlOpt}

according to nonlinear least squares error, maximum improvement factor and number of QN iterations via simulation and convergence diagrams. Moreover, we present the tables of our simulation results for additional information in Appendix A.

2.1 Parameter Optimization Algorithm for Dynamical System

Parameter vector optimization in a mathematical model coming from asset flow theory is a challenging inverse problem. Using hybrid of hyperbox and multi-start methods, we examine the effectiveness of grid and random approaches to obtain suitable parameters like q_1, c_1, q_2 and c_2. The main structure of the algorithm with nested function call is represented in Algorithm 1. Table 2.1 and Table 2.2 show the description of the input and the output variables, respectively.

Hyperbox is a non-degenerate closed interval (see [30]). Let $\mathbf{l} = (l_1, l_2, ..., l_n)$ and $\mathbf{u} = (u_1, u_2, ..., u_n)$ be two points in $\mathbb{R}^n$ with $l_i \leq u_i$ for $i = 1, 2, ..., n$. The n-dimensional hyperbox is a set $\mathbb{X} = \{\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n : l_i \leq x_i \leq u_i \; and \; l_i \neq u_i, \forall \; i \in \mathbb{N}_n\}$ for a given $\mathbf{l}$ and $\mathbf{u}$ points. In our study, we use this set for parameter vector optimization application to set the boundaries according to expert opinion. Numerical optimization algorithms generally start from an initial point and try to converge the minimum or

Algorithm 1 Parameter vector optimization algorithm on a single event.

▷ Look at Table 2.1 and Table 2.2 for the description of the input and the output variables, respectively.
function OPT$(\mathbb{K}, MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2)$
    Set time indices $t_s = [i, i+n-1]$ for $i^{th}event$
    Set $x_2 = 0.5, x_3 = 0$ and $x_4 = 0$
    Initialize $A = [], \hat{K} = [], \hat{c}_{locOpt} = [], \hat{MIF}_{locOpt} = []$ and $\hat{QNiter}_{locOpt} = []$
    $\hat{MP} = MP[i-m : i+n-1]$
    $\hat{NAV} = NAV[i-m : i+n-1]$
    $\tilde{MP} = \hat{MP}[m+1 : m+n]$
    $A = \mathbf{zeros}(n, 1)$
    ▷ Relative valuation change loop
    for $s = 1 : n$ **do**
        ▷ Chronic discount loop
        for $k = 1 : m$ **do**
            $u = s + m - k$
            $A[s] = A[s] + (\hat{NAV}[u] - \hat{MP}[u])/\hat{NAV}[u]e^{0.25k}$
        end for
        $A[s] = (\hat{NAV}[s+m] - \hat{MP}[s+m])/\hat{NAV}[s+m] - A[s]/3.23180584357794$
    end for
    ▷ Multi-start initial parameter loop
    for $j = 1 : \mathbf{length}(\mathbb{K})$ **do**
        $\tilde{K}_j = \mathbb{K}[j, :]$
        ▷ The dynamical system is solved via Runge-Kutta (RK4) method. Then
        ▷ the function and its gradient are evaluated as a nested call at each QN
        ▷ iteration.
        $[\bar{K}, QNiter, success] = \mathbf{QN}(\tilde{K}_j, t_s, h_{RK4}, A, \tilde{MP}, x_2, x_3, x_4, \varepsilon_1)$
        if *success* **then**
            $c_{NLS} = \mathbf{NLS}(\bar{K}, t_s, h_{RK4}, A, \tilde{MP}, x_2, x_3, x_4)$
            if $(c_{NLS} < \varepsilon_2) \& (\bar{K} \geq 0)$ **then**
                $c_{NLS_{init}} = \mathbf{NLS}(\tilde{K}_j, t_s, h_{RK4}, A, \tilde{MP}, x_2, x_3, x_4)$
                Append $\bar{K}$ to $\hat{K}$
                Append c_{NLS} to $\hat{c}_{locOpt}$
                Append $(c_{NLS}/c_{NLS_{init}})$ to $\hat{MIF}_{locOpt}$
                Append $QNiter$ to $\hat{QNiter}_{locOpt}$
            end if
        end if
    end for
    $c_{iGlOpt} = \mathbf{min}(\hat{c}_{locOpt})$
    $j_{iGlOpt} = \mathbf{find}(\hat{c}_{locOpt} == c_{iGlOpt})$
    $K_{iGlOpt} = \hat{K}[j_{iGlOpt}, :]$
    $MIF_{iGlOpt} = \hat{MIF}_{locOpt}[j_{iGlOpt}, :]$
    $QNiter_{iGlOpt} = \hat{QNiter}_{locOpt}[j_{iGlOpt}, :]$
end function

maximum value which is under local or global space, considering the mathematical metrics. The selection of the initial values in its large search space with an appropriate method is an important issue. For a dynamical system we examine the effect of the two different types of the initial parameter vector pool with attention to aspects such as selection the initial vector for getting reasonable solution in an efficient manner and methods using Monte Carlo simulation. In this study, we use two types of approaches including grid approach and random approach to generate initial parameter vector pools for our problem and compare them, unlike [4].

In the grid approach, we divide the given hyperbox space $\mathbb{X}$ into grid along its dimensions and the set of grid points are chosen as parameter vector. For example, given the rectangle $[l_1,u_1]\times[l_2,u_2]$ of 2-dimensional hyperbox, and number of the grid points n_1 and n_2 for each two dimensions, we obtain totaly $n_1\times n_2$ grid points after n_1-1 and n_2-1 division along each dimension. In our optimization problem, we use an initial parameter vector $K=(c_1,q_1,c_2,q_2)\in\mathbb{R}^4_+$. Let $[l_1,u_1]\times[l_2,u_2]\times[l_3,u_3]\times[l_4,u_4]$ be the 4-dimensional hyperbox, and n_1, n_2, n_3, and n_4 be number of the grid points for the corresponding dimensions respectively. Therefore, we obtain totaly $n_1\times n_2\times n_3\times n_4$ grid points after n_1-1, n_2-1, n_3-1, and n_4-1 division along each dimension for our problem. Thus, they are perfectly uniformly distributed in the hyperbox.

In the random approach, the points are chosen according to uniform distribution of the numbers in each dimension. For example, given the rectangle $[l_1,u_1]\times[l_2,u_2]$ of 2-dimensional hyperbox, we generate two uniformly distributed random numbers r_1 and r_2 in the interval $(0,1)$ using Matlab rand function (see [31]) and then stretch it to hyperbox interval $[\mathbf{l},\mathbf{u}]$ as $((u_1-l_1)r_1+l_1,\ (u_2-l_2)r_2+l_2)$ for the initial parameter. Similarly, for our optimization problem, we obtain hyperbox interval $[\mathbf{l},\mathbf{u}]$ as $((u_1-l_1)r_1+l_1,\ (u_2-l_2)r_2+l_2,(u_3-l_3)r_3+l_3,(u_4-l_4)r_4+l_4)$. We repeat this process for each size of the initial parameter pool $\mathbb{K}$ for both approaches.

Closed-end funds (CEFs) trade on the New York Stock Exchange (NYSE) and a fund may trade on lower than the net asset value (NAV) (called a discount) or higher (called a premium) than the NAV with lots of challenges [32]. Given an n-day

period of market price (MP) and net asset value (NAV) pair, we compute optimal parameter vector $\bar{K} = (c_1, q_1, c_2, q_2) \in \mathbb{R}^4_+$ for related period using initial parameter vector pool $\mathbb{K}$. In this study, we use proxy for time series of MP and NAV via simulation, unlike [4]. We continue this process for overlapping periods starting i-th and $i+1$-th days respectively throughout the time series of MP and NAV. For each selected event and initial parameter vector, we use quasi-Newton (QN) method having the Broyden-Fletcher-Goldfarb-Shanno (BFGS) formula with backtracking line search algorithm to optimize the function $F[\tilde{K}]$ where $F[\tilde{K}]$ represents the sum of exponentially weighted squared differences between the proxy for actual MP values via simulation and the computed MP values. $\tilde{K} \in \mathbb{K}$ is an initial or a candidate parameter vector.

2.1.1 The dynamical system of asset flow differential equations in matrix form

In this study, we use the following 3rd version of AFDEs in Eq. (2.1) in the matrix form with problem constraints in [7] and [8]. Duran [8] studied the stability analysis of the solutions for the dynamical system of nonlinear AFDEs in $\mathbb{R}^4$ analytically and numerically.

$$\begin{bmatrix} \frac{1}{x_1} & 0 & 0 & 0 \\ \frac{-x_2(1-x_2)}{x_1} & 1 & 0 & 0 \\ -c_1 q_1 \frac{1}{x_1} & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \end{bmatrix} = \begin{bmatrix} \delta log(\frac{k}{1-k}\frac{1-x_2}{x_2}) \\ k(1-x_2) - (1-k)x_2 \\ -c_1 x_3 \\ c_2 \left(q_2 \left(\frac{P_a - x_1}{P_a} - D(x_1(t-1), P_a(t-1), ..., x_1(t-n), P_a(t-n)) \right) - x_4 \right) \end{bmatrix} \tag{2.1}$$

The corresponding variables and parameters are as follows:

$x_1(t)$: The market price of the single asset at time t.

$\frac{1}{x_1(t)} x_1'(t)$: The relative price change.

$P_a(t)$: The fundamental value.

$V(t)$: The net asset value price at time t where $V(t)$ can be taken as a proxy for $P_a(t)$ in practice.

$x_2(t)$: The fraction of total funds in the asset.

$x_3(t)$: The trend-based component of the investor preference.

$x_4(t)$: The value-based component of the investor preference.

$k(t)$: The transition rate.

$K = (c_1, q_1, c_2, q_2) \in \mathbb{R}^4_+$: The parameter vector for which we seek optimal/ feasible values.

$D(x_1(t-1), P_a(t-1), x_1(t-2), P_a(t-2), ..., x_1(t-n), P_a(t-n))$: The chronic discount amount measured approximately over the previous few finite $n \geq 1$ days.

The constants δ, $\frac{1}{c_1}$ and $\frac{1}{c_2}$ are the time scales for the price equation, the momentum and valuation investment strategies, respectively. δ can be taken as 1.

Equation (2.1) can be rewritten in the form of

$$U^{'} = \mathbf{f}(U, K, P_a)$$

similar to that of Duran [21] using the inverse of real function valued triangular matrix where

$$U = [x_1\ x_2\ x_3\ x_4]^T,\ \ U^{'} = dU/dt,$$

and

$$\mathbf{f} = [f_1\ f_2\ f_3\ f_4]^T.$$

$$x_1^{'} = \frac{\delta x_1(1-2x_2+x_3+x_4)}{(1-x_3-x_4)x_2} \tag{2.2}$$

$$x_2^{'} = \frac{(1-2x_2+x_3+x_4)(1+\delta(1-x_2)-x_3-x_4)}{1-x_3-x_4} \tag{2.3}$$

$$x_3^{'} = c_1\frac{q_1\delta(1-2x_2+x_3+x_4)-(1-x_3-x_4)x_2x_3}{(1-x_3-x_4)x_2} \tag{2.4}$$

$$x_4^{'} = c_2\frac{q_2P_a - q_2x_1 - P_ax_4}{P_a} \tag{2.5}$$

$$U^{'} = \mathbf{f}(U, K, P_a)\ \ and\ \ U(t_1) = [\bar{S}(t_1)\ 0.5\ 0\ 0]^T \tag{2.6}$$

We solve the IVP (2.6) above for U by using Runge$-$Kutta (RK4) method and an assumed value $\tilde{K}$ from the initial parameter pool $\mathbb{K}$.

2.1.2 The nonlinear least squares technique for the optimization problem

We use nonlinear least squares (NLS) technique with initial value problem having AFDEs as in [4]. We define $F[\tilde{K}]$ such that

$$F[\tilde{K}] := \sum_{s=i}^{i+n-1} W(s-i+1)\{\bar{S}(t_s) - x_1(\tilde{K}, t_s)\}^2, \tag{2.7}$$

where $F[\tilde{K}]$ represents the sum of exponentially weighted squared differences between the simulated MP values $\bar{S}(t_s)$ and the computed MP values $x_1(\tilde{K}, t_s)$ obtained from the first row vector of the numerical solution U of IVP (2.6) by picking the values at time t_s where $t_s \in [i, i+n-1]$ for i^{th} parameter vector. For example,

$$W = (0.114051, 0.146444, 0.188038, 0.241445, 0.310022)^T$$

for $n = 5$.

We try to minimize $F[\tilde{K}]$ over $\mathbb{R}^4_+$ by using line search algorithm.

2.2 Convergence of The Nonlinear Least Squares Technique for The Differential Equations

Theorem 2.2.1. *Assume that F is convex, twice differentiable function with domain having $\mathbb{R}^4_+$. Additionally*

i) *∇F is Lipschitz with parameter L,*

ii) *F is strongly convex with parameter m,*

iii) *$\nabla^2 F$ is Lipschitz with parameter M.*

If F is strongly convex, then quasi-Newton method using BFGS formula with backtracking line search converges globally from any initial parameter vector $\tilde{K}_0$ and initial positive definite matrix H_0. Moreover, if F is strongly convex and $\nabla^2 F$ is Lipschitz continuous, then local convergence is superlinear, that is, for all $k \geq k_0$,

$$\|\tilde{K}_k - \bar{K}_\star\|_2 \leq c_k \|\tilde{K}_{k-1} - \bar{K}_\star\|_2 \tag{2.8}$$

for sufficiently large k where $c_k \to 0$, k_0 and c_k depend on L, m, and M.

Proof. See [33].

See [34], [33], [35] and [36] for a comprehensive details of quasi-Newton methods.

We do not have convexity assumption for the objective function F. We perform Monte Carlo simulation and obtain convergence diagrams for the challenging unconstrained optimization problem. In general, the reason why Monte Carlo simulation works is the Law of Large Numbers (see [37] and [38]).

Theorem 2.2.2. *(Strong Law of Large Numbers) Let X_1, X_2, . . . , X_j, . . . be a sequence of independent and identically distributed random variables, with a finite expected value $E[X_j] = \mu$. Then, with probability 1,*

$$\frac{1}{N}\sum_{j=1}^{N} X_j \ \rightarrow \mu \ a.s. \ as \ N \rightarrow \infty. \tag{2.9}$$

Proof. The general form of the strong law was proved by the mathematician A. N. Kolmogorov. See [39].

Corollary 2.2.1 shows the case of the Theorem 2.2.2 for the nonlinear least squares (NLS) error. Also, Corollary 2.2.2 and Corollary 2.2.3 show the implementation of the law of large numbers for the maximum improvement factor (MIF) and quasi-Newton (QN) iterations, respectively.

Corollary 2.2.1. *Let F_j be the average NLS error over $L-15$ events for the time series of length L at $j'th$ iteration and F_1, F_2, . . . , F_j, . . . be a sequence of independent and identically distributed random variables with finite expected value $E[F_j] = \nu$. Then*

$$\frac{1}{N}\sum_{j=1}^{N} F_j \ \rightarrow \nu \ a.s. \ as \ N \rightarrow \infty \tag{2.10}$$

where N is the length of the initial parameter vector pool.

Corollary 2.2.2. *Let G_j be the average maximum improvement factor (MIF) over $L-15$ events for the time series of length L at $j'th$ iteration and G_1, G_2, . . . , G_j, . . . be a sequence of independent and identically distributed random variables with finite expected value $E[G_j] = \lambda$. Then*

$$\frac{1}{N}\sum_{j=1}^{N} G_j \ \rightarrow \lambda \ a.s. \ as \ N \rightarrow \infty \tag{2.11}$$

where N is the length of the initial parameter vector pool.

Corollary 2.2.3. *Let H_j be the average quasi-Newton (QN) iteration number over $L-15$ events for the time series of length L at $j'th$ iteration and H_1, H_2, . . . , H_j, . . .*

Table 2.3 : Tuning parameters for optimization algorithms.

Event period	5
Runge-Kutta (RK4) method step size	0.05
Threshold for the gradient	10^{-5}
Threshold for the nonlinear least squares error	0.16

be a sequence of independent and identically distributed random variables with finite expected value $E[H_j] = \theta$. *Then*

$$\frac{1}{N}\sum_{j=1}^{N} H_j \ \rightarrow \theta \ a.s. \ as \ N \rightarrow \infty \tag{2.12}$$

where N is the length of the initial parameter vector pool.

2.3 The Experimental Design

In the parameter optimization algorithm, we use the BFGS update formula (see [33,40,41]) to solve the optimization problem. Table 2.3 displays the experimental design and threshold values for the optimization in Algorithms 1, 2 and 3. We try to find the parameter giving the minimum error with using the initial parameter pool $\mathbb{K}$ for each event. We start to run quasi-Newton with any given initial parameter vector. This process continues until we process all parameter vectors from the initial parameter vector pool for any given event. We permit only the positive candidate parameter vectors satisfying the threshold condition for the nonlinear least squares error.

We generate two different types of initial parameter vector pools which include starting parameter vectors. They are $\mathbb{K}_g$ for grid approach and $\mathbb{K}_r$ for random approach. For these two types, parameters are selected from a bounded hyperbox search space. Lower and upper bounds are determined according to the previous studies (see [4,29]). Table 2.4 displays the values of these bounds. While we determine the values on the grid points of the bounded real-number space of hyperbox for grid approach, we select values randomly from that space for the random approach.

In this study, we use time series of market price and net asset value data obtained via our simulation rather than real closed-end fund data, because we may consider more possible cases via simulation than that of real data for a particular time interval. Using random-walk, we generate 40 time series described in Table 2.5 and Table 2.6 whose

Table 2.4 : Upper and lower bounds of the initial parameters.

	Lower Bound	Upper Bound
c_1	0.001	1.1
q_1	1.1	100.1
c_2	0.005	1.1
q_2	0.01	50.1

lengths are 1500 business days, approximately six years having daily closing prices. They mimic several long term real market scenarios. Since time series are handled as moving overlapped 5 day event periods and the algorithm needs to calculate the chronic discount over the past 10 days, we try to optimize 1485 time period for each time series. Thus, we have $(40 * 1485) = 59400$ events having local market scenarios such as chronic discount, chronic premium, getting bigger difference between MP and NAV, getting smaller difference between MP and NAV or crossing-over behavior over 5-day time periods.

Moreover, we generate the initial parameter vector pools $\mathbb{K}_g$ and $\mathbb{K}_r$ whose lengths are 256, 512, 1024, 2048 and 4096 using the two approaches including grid approach and random approach. So we test each time series with 10 parameter vector pools.

Furthermore, we use the first four central moments which are the important properties for decision-making process while generating our test cases. The usage of these properties can be seen on the article about the profitable trading and risk management strategy by Duran and Bommarito [42]. The first four moment includes the values of the mean, standard deviation, skewness and kurtosis of the time series. They are considered to find the hot spot points of the algorithm and to improve it.

Table 2.5 : Statistical properties of the time series Dsc.

Price time series (PTS)	Mean (MP / NAV)	Volatility ratio (MP / NAV)	Skewness (MP / NAV)	Kurtosis (MP / NAV)	MP range [Min - Max]	NAV range [Min - Max]	Status (Initial / Final)
Dsc_01	54.38 / 64.74	3.82 / 5.74	0.54 / -0.17	2.40 / 1.65	48.54 - 64.98	54.60 - 75.55	D / D
Dsc_02	70.78 / 58.57	6.72 / 2.42	-0.02 / -0.29	2.10 / 2.93	57.68 - 83.34	51.78 - 63.96	D / P
Dsc_03	51.71 / 62.11	4.07 / 2.73	0.21 / 0.53	1.63 / 2.34	44.69 - 59.08	57.65 - 69.31	D / D
Dsc_04	51.95 / 64.52	3.74 / 3.52	-0.09 / -0.37	3.01 / 2.06	43.15 - 62.27	56.94 - 71.00	D / D
Dsc_05	53.21 / 58.97	6.02 / 2.60	-0.2 / -0.35	1.75 / 2.05	41.62 - 64.00	52.35 - 64.38	D / D
Dsc_06	46.73 / 61.04	7.73 / 5.48	-0.03 / 0.70	1.51 / 2.32	33.44 - 59.58	52.22 - 74.66	D / D
Dsc_07	67.41 / 58.64	6.73 / 2.29	0.30 / 0.61	2.34 / 2.47	56.07 - 84.62	54.98 - 65.48	D / P
Dsc_08	74.19 / 58.96	8.57 / 3.04	0.11 / 0.01	2.38 / 2.12	58.10 - 93.92	52.54 - 65.85	D / P
Dsc_09	71.95 / 60.33	10.78 / 5.10	1.13 / 0.37	3.21 / 1.62	56.77 - 98.79	53.36 - 70.96	D / P
Dsc_10	62.39 / 64.83	3.71 / 3.41	0.56 / -0.60	2.50 / 2.59	54.95 - 71.92	55.98 - 70.79	D / P
Dsc_11	64.06 / 58.73	4.07 / 2.65	-0.14 / -0.15	2.34 / 2.66	55.19 - 72.82	51.59 - 65.21	D / P
Dsc_12	74.65 / 66.60	7.73 / 5.53	-1.21 / 0.09	2.97 / 2.37	56.05 - 83.39	54.53 - 80.69	D / P
Dsc_13	68.4 / 56.29	9.12 / 3.12	-0.22 / -0.62	1.44 / 2.42	54.07 - 81.61	48.97 - 61.02	D / P
Dsc_14	84.87 / 62.28	18.43 / 3.54	0.25 / -0.42	1.64 / 1.95	58.87 - 120.47	54.23 - 68.61	D / P
Dsc_15	79.09 / 54.51	12.97 / 2.98	0.72 / 0.15	2.21 / 1.59	59.00 - 108.62	49.45 - 60.19	D / P
Dsc_16	44.88 / 60.57	8.36 / 3.74	0.85 / 0.13	2.29 / 1.79	35.08 - 63.09	54.16 - 68.27	D / D
Dsc_17	50.21 / 55.45	6.05 / 2.15	0.63 / 0.45	2.36 / 2.70	41.30 - 65.31	50.99 - 62.00	D / D
Dsc_18	52.55 / 61.10	8.31 / 2.23	0.17 / 0.11	1.32 / 1.87	41.22 - 65.63	56.38 - 65.89	D / D
Dsc_19	57.64 / 59.63	2.06 / 1.83	0.37 / 0.16	2.83 / 2.26	52.54 - 63.31	55.19 - 63.84	D / P
Dsc_20	58.59 / 62.64	2.83 / 2.49	0.79 / 0.41	2.67 / 2.59	53.84 - 66.41	56.89 - 68.67	D / D

Table 2.6 : Statistical properties of the time series Prm.

Price time series (PTS)	Mean (MP / NAV)	Volatility ratio (MP / NAV)	Skewness (MP / NAV)	Kurtosis (MP / NAV)	MP range [Min - Max]	NAV range [Min - Max]	Status (Initial / Final)
Prm_01	56.22 / 64.74	3.95 / 5.74	0.54 / -0.17	2.40 / 1.65	50.19 - 67.19	54.60 - 75.55	P / D
Prm_02	55.04 / 59.26	3.80 / 2.54	0.55 / -0.40	2.23 / 2.46	48.43 - 63.36	52.35 - 64.67	P / D
Prm_03	53.71 / 64.52	3.86 / 3.52	-0.09 / -0.37	3.01 / 2.06	44.61 - 64.38	56.94 - 71.00	P / D
Prm_04	55.01 / 58.97	6.22 / 2.60	-0.20 / -0.35	1.75 / 2.05	43.03 - 66.17	52.35 - 64.38	P / D
Prm_05	76.71 / 58.96	8.86 / 3.04	0.11 / 0.01	2.38 / 2.12	60.07 - 97.11	52.54 - 65.85	P / P
Prm_06	55.00 / 62.14	8.00 / 2.41	-0.28 / 0.17	1.48 / 2.18	42.12 - 67.36	57.29 - 67.80	P / D
Prm_07	74.39 / 60.33	11.14 / 5.10	1.13 / 0.37	3.21 / 1.62	58.70 - 102.14	53.36 - 70.96	P / P
Prm_08	65.89 / 54.89	8.40 / 2.13	0.31 / 0.28	1.46 / 2.00	55.13 - 81.37	50.90 - 60.02	P / P
Prm_09	67.91 / 62.86	4.45 / 1.92	-0.82 / 0.30	3.13 / 2.59	56.71 - 77.34	58.00 - 67.87	P / P
Prm_10	57.73 / 69.33	7.50 / 5.64	0.28 / 0.39	1.79 / 1.96	46.51 - 73.66	59.90 - 82.69	P / D
Prm_11	50.45 / 68.58	6.49 / 2.60	0.77 / -0.15	1.92 / 2.42	43.51 - 64.41	60.00 - 74.87	P / D
Prm_12	65.46 / 64.73	7.75 / 5.27	0.48 / 0.07	1.86 / 1.56	52.06 - 80.25	56.15 - 74.62	P / P
Prm_13	70.73 / 59.24	6.41 / 2.85	0.38 / 0.40	2.80 / 3.17	58.03 - 87.80	53.42 - 68.06	P / P
Prm_14	48.97 / 60.55	4.06 / 2.15	0.47 / -0.45	3.86 / 3.12	40.76 - 61.39	55.01 - 65.74	P / D
Prm_15	87.75 / 62.28	19.06 / 3.54	0.25 / -0.42	1.64 / 1.95	60.87 - 124.56	54.23 - 68.61	P / P
Prm_16	46.40 / 60.57	8.64 / 3.74	0.85 / 0.13	2.29 / 1.79	36.27 - 65.23	54.16 - 68.27	P / D
Prm_17	88.69 / 62.19	15.11 / 3.61	-0.49 / 0.49	1.80 / 2.65	59.58 - 111.08	54.82 - 70.88	P / P
Prm_18	61.40 / 52.04	3.91 / 3.06	-0.02 / 0.22	2.06 / 3.10	53.36 - 69.19	45.25 - 60.22	P / P
Prm_19	67.04 / 59.25	2.88 / 1.61	0.04 / 0.29	2.55 / 3.04	60.65 - 74.26	55.16 - 64.18	P / P
Prm_20	58.74 / 61.01	2.85 / 1.83	0.11 / 0.44	2.61 / 2.37	52.47 - 66.85	57.12 - 66.05	P / D

Table 2.5 and Table 2.6 describe the market price and net asset value time series and their statistical properties like volatility behavior and ranges for the time series group Dsc for discount and Prm for premium situation of market price with respect to the corresponding net asset value based on the initial status, respectively. Considering the different statistical properties of the parameter vector pools $\mathbb{K}_g$ and $\mathbb{K}_r$ we aim to scan and represent the bounded space as much as possible with satisfied sample size. To check the representation success of the parameter pool $\mathbb{K}$ we apply the principal component analysis (PCA) which is preferred to detect the pattern of the dominant features (see [43]). Four moments, minimum and maximum values for MP, and NAV are used as an input feature. Also as a binary feature we use the discount and premium status as 1 and 0, respectively. Thus we project the high dimensional feature space into 2D and 3D spaces using PCA. In Figure 2.2 and Figure 2.3, we show the dominant features and it is seen that there is a general sample distribution to handle many points of the space. Here, the 40 points in the figures generated by using the 40 time series are represented via different colors so that each point can be visible at right position.

2.4 Initial Parameter Vector Pool Selection Results and Convergence Diagrams

We examine the effects of the grid and random approaches for the selection of the initial parameter pool $\mathbb{K}$ with 59400 events for the numerical optimization problem of the dynamical system. We compare the efficiency of two approaches with respect to nonlinear least squares (NLS) error, maximum improvement factor (MIF) and number of QN iterations. Smaller error for nonlinear least squares method, smaller MIF value for the computing duration or smaller iteration number for quasi-Newton algorithm may be considered as better performance. Algorithms 2 and 3 show the pseudo codes of the Monte Carlo simulation with the generation of the initial parameter pools $\mathbb{K}_g$ and $\mathbb{K}_r$, respectively.

2.4.1 The Comparison of the two approaches according to NLS errors

Table A.1 and Table A.2 show the converged average nonlinear least squares (NLS) error values in order to compare the grid and random approaches via simulation results for Dsc and Prm time series groups respectively. Figure 2.4 obtained via Algorithm

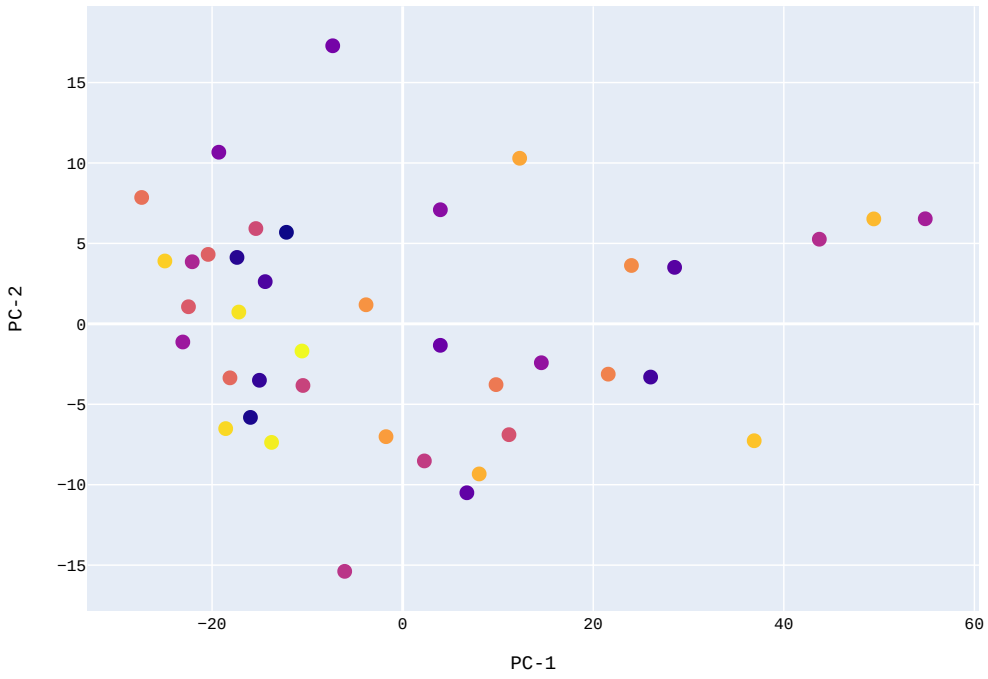

Figure 2.2 : The projection of the high dimensional feature space into 2D space using principal component analysis where PC-*i* corresponds to the *i*-th largest eigenvalue.

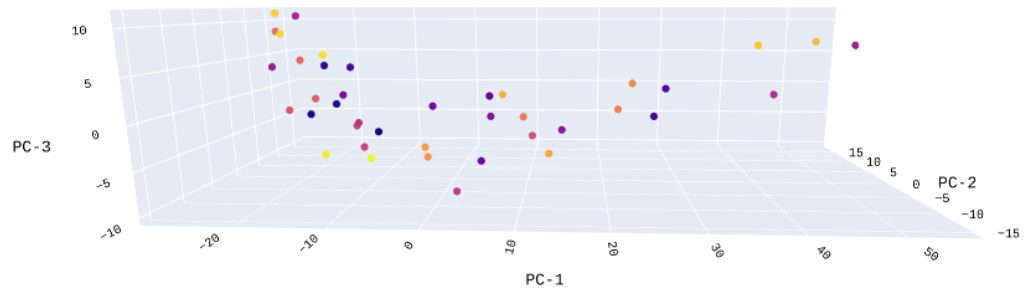

Figure 2.3 : The projection of the high dimensional feature space into 3D space using principal component analysis where PC-*i* corresponds to the *i*-th largest eigenvalue.

Algorithm 2 Monte Carlo simulation via grid approach.

Inputs:
$MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2$
N: Size of the initial parameter vector pool
$[l_1, l_2, l_3, l_4]$: Lower bounds for each dimension
$[u_1, u_2, u_3, u_4]$: Upper bounds for each dimension
$[n_1, n_2, n_3, n_4]$: Number of grid points for each dimension

Output:
c_{MK}: Sequence of Monte Carlo iterations for NLS error
K_{MK}: Sequence of Monte Carlo iterations for K parameter vector
MIF_{MK}: Sequence of Monte Carlo iterations for MIF
$QNiter_{MK}$: Sequence of Monte Carlo iterations for QN iteration
▷ Generates n_1 grid points with equally gap size $h_1 = (u_1 - l_1)/(n_1 - 1)$
$c_1 = \textbf{linspace}(l_1, u_1, n_1)$
$q_1 = \textbf{linspace}(l_2, u_2, n_2)$
$c_2 = \textbf{linspace}(l_3, u_3, n_3)$
$q_2 = \textbf{linspace}(l_4, u_4, n_4)$
$N = n_1 * n_2 * n_3 * n_4$
$\mathbb{K}_g = [c_1\ q_1\ c_2\ q_2]$

function SIMG$(\mathbb{K}_g, MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2)$
    Initialize $\hat{c}_{GlOpt} = [], \hat{c}_{MK} = [], \hat{K}_{GlOpt} = []$ and $\hat{K}_{MK} = []$
    Initialize $\hat{MIF}_{GlOpt} = [], \hat{MIF}_{MK} = [], \hat{QNiter}_{GlOpt} = []$ and $\hat{QNiter}_{MK} = []$
    ▷ First event index
    $i_{first} = m + 1$
    ▷ Last event index
    $i_{last} = \textbf{length}(MP) - n$
    $c_{sum} = 0, K_{sum} = 0, MIF_{sum} = 0, QNiter_{sum} = 0$
    ▷ Iteration over parameter vectors
    for $j = 1 : \textbf{length}(\mathbb{K}_g)$ **do**
        for $i = i_{first} : i_{last}$ **do**
            $[\hat{c}_{GlOpt}[i], \hat{K}_{GlOpt}[i], \hat{MIF}_{GlOpt}[i], \hat{QNiter}_{GlOpt}[i]] = \textbf{OPT}(\mathbb{K}_g[1 : j, :], MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2)$
        end for
        $c_{sum} = c_{sum} + \textbf{average}(\hat{c}_{GlOpt})$
        $c_{MK}[j] = c_{sum}/j$

        $K_{sum} = K_{sum} + \textbf{average}(\hat{K}_{GlOpt})$
        $K_{MK}[j] = K_{sum}/j$

        $MIF_{sum} = MIF_{sum} + \textbf{average}(\hat{MIF}_{GlOpt})$
        $MIF_{MK}[j] = MIF_{sum}/j$

        $QNiter_{sum} = QNiter_{sum} + \textbf{average}(\hat{QNiter}_{GlOpt})$
        $QNiter_{MK}[j] = QNiter_{sum}/j$
    end for
end function

Algorithm 3 Monte Carlo simulation via random approach.

Inputs:
$MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2$
N: Size of the initial parameter vector pool
$[l_1, l_2, l_3, l_4]$: Lower bounds for each dimension
$[u_1, u_2, u_3, u_4]$: Upper bounds for each dimension

Output:
c_{MK}: Sequence of Monte Carlo iterations for NLS error
K_{MK}: Sequence of Monte Carlo iterations for K parameter vector
MIF_{MK}: Sequence of Monte Carlo iterations for MIF
$QNiter_{MK}$: Sequence of Monte Carlo iterations for QN iteration

$c_1 = (u_1 - l_1).* \mathbf{rand}(N,1) + l_1$
$q_1 = (u_2 - l_2).* \mathbf{rand}(N,1) + l_2$
$c_2 = (u_3 - l_3).* \mathbf{rand}(N,1) + l_3$
$q_2 = (u_4 - l_4).* \mathbf{rand}(N,1) + l_4$
$\mathbb{K}_r = [c_1\ q_1\ c_2\ q_2]$

function SIMR$(\mathbb{K}_r, MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2)$
    Initialize $\hat{c}_{GlOpt} = [], \hat{c}_{MK} = [], \hat{K}_{GlOpt} = []$ and $\hat{K}_{MK} = []$
    Initialize $\hat{MIF}_{GlOpt} = [], \hat{MIF}_{MK} = [], \hat{QNiter}_{GlOpt} = []$ and $\hat{QNiter}_{MK} = []$
    ▷ First event index
    $i_{first} = m + 1$
    ▷ Last event index
    $i_{last} = \mathbf{length}(MP) - n$
    $c_{sum} = 0, K_{sum} = 0, MIF_{sum} = 0, QNiter_{sum} = 0$
    ▷ Iteration over parameter vectors
    for $j = 1 : \mathbf{length}(\mathbb{K}_r)$ **do**
        for $i = i_{first} : i_{last}$ **do**
            $[\hat{c}_{GlOpt}[i], \hat{K}_{GlOpt}[i], \hat{MIF}_{GlOpt}[i], \hat{QNiter}_{GlOpt}[i]] = \mathbf{OPT}(\mathbb{K}_r[1 : j, :], MP, NAV, i, n, m, h_{RK4}, \varepsilon_1, \varepsilon_2)$
        end for
        $c_{sum} = c_{sum} + \mathbf{average}(\hat{c}_{GlOpt})$
        $c_{MK}[j] = c_{sum}/j$

        $K_{sum} = K_{sum} + \mathbf{average}(\hat{K}_{GlOpt})$
        $K_{MK}[j] = K_{sum}/j$

        $MIF_{sum} = MIF_{sum} + \mathbf{average}(\hat{MIF}_{GlOpt})$
        $MIF_{MK}[j] = MIF_{sum}/j$

        $QNiter_{sum} = QNiter_{sum} + \mathbf{average}(\hat{QNiter}_{GlOpt})$
        $QNiter_{MK}[j] = QNiter_{sum}/j$
    end for
end function

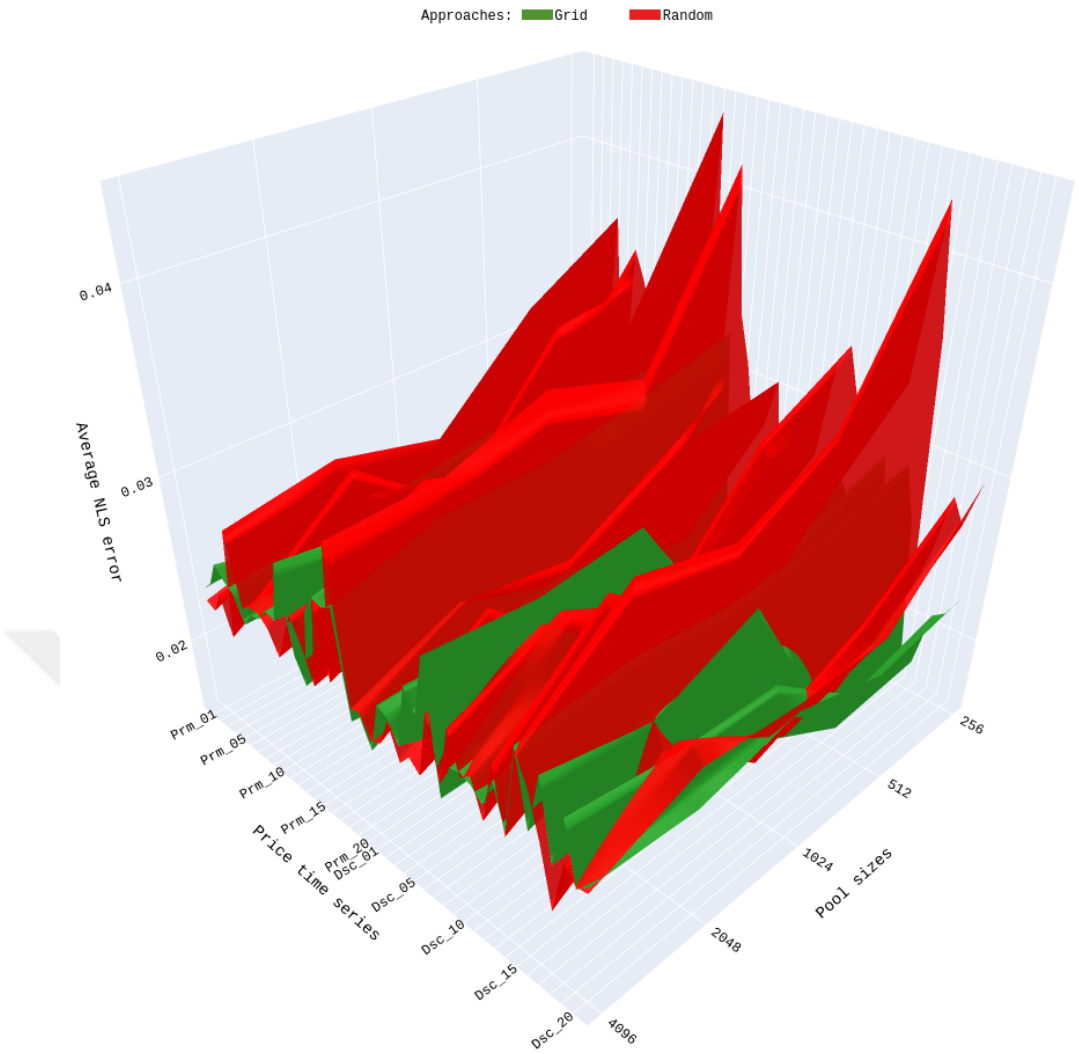

Figure 2.4 : Comparison of the grid and random approaches for all time series in the dataset according to average NLS error.

1 from the initial parameter pools $\mathbb{K}_g$ and $\mathbb{K}_r$, displays the dependence of the average NLS errors of the two approaches on the size of initial parameter pool for 40 time series based on Table A.1 and Table A.2. The averaged NLS errors via random approach are higher than that of the grid approach mostly. Moreover, the averaged NLS error for the random approach decreases as the size of initial parameter pool increases. They are valuable results for the nonlinear optimization problem.

Based on the volatility behavior described in Table 2.5 and Table 2.6, we see that generally the NLS error is bigger for the time series pair as proxy to MP and NAV whose volatilities are sufficiently larger for both MP and NAV provided that the other variables are unchanged. This is consistent with the results in [10].

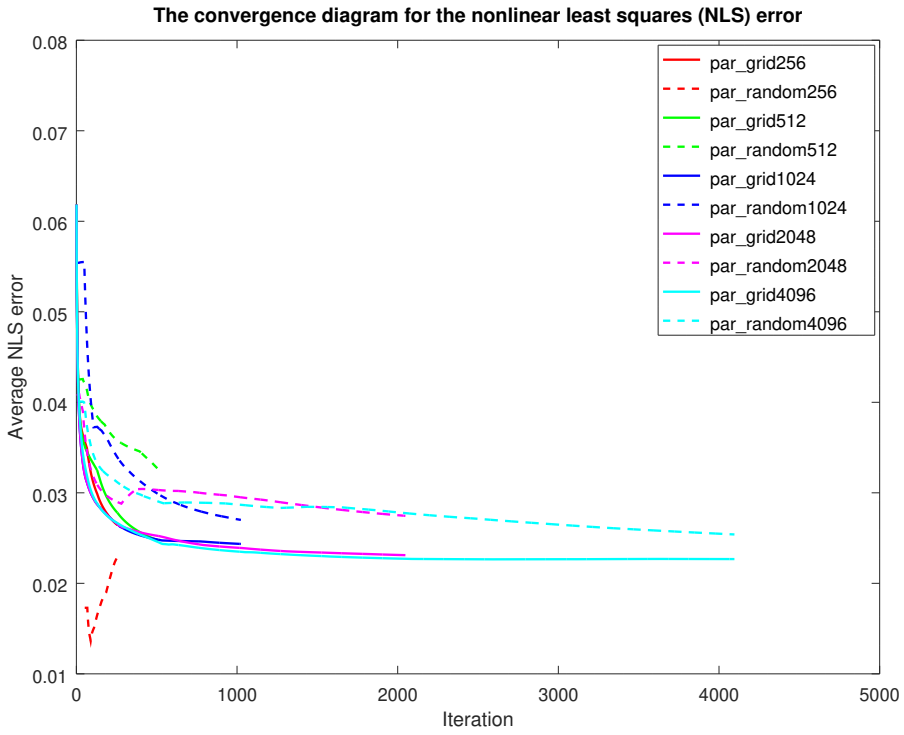

Figure 2.5 : Monte Carlo simulation of the NLS error for curve fitting of Dsc_20 for each approach.

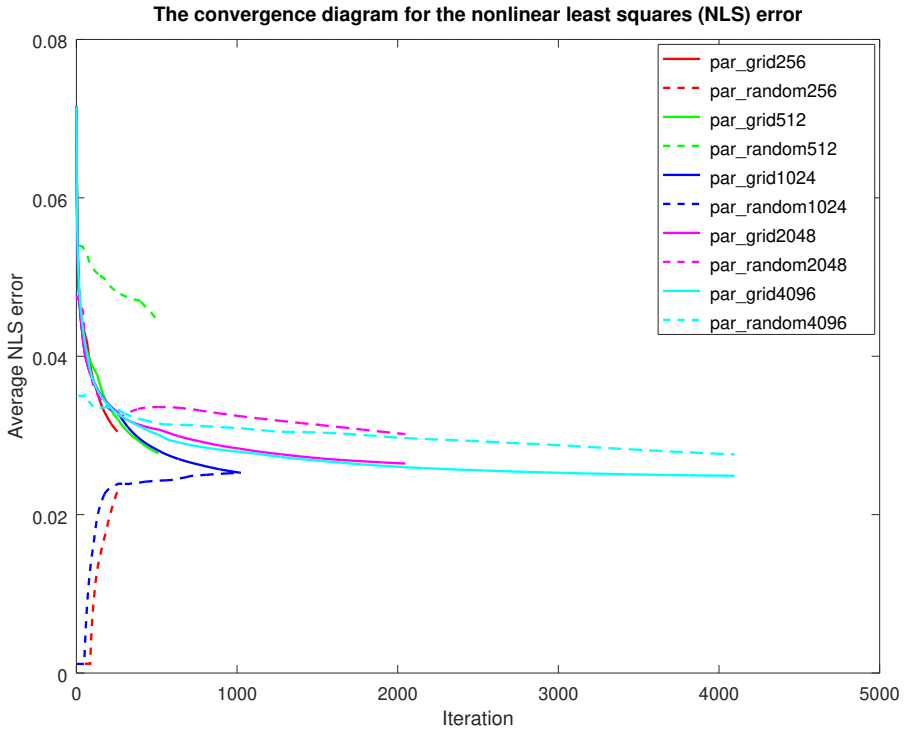

Figure 2.6 : Monte Carlo simulation of the NLS error for curve fitting of Prm_08 for each approach.

Table 2.7 : The number of successful approaches with respect to NLS error among the Dsc and Prm time series for the pools with different sizes.

Sequence type	Parameter type	Different parameter vector lengths					
		256	512	1024	2048	4096	Total
Dsc	grid	0	10	2	0	3	15
	random	0	0	1	0	4	5
Prm	grid	0	12	1	0	4	17
	random	0	0	1	0	2	3
Total	grid	0	22	3	0	7	32
	random	0	0	2	0	6	8

Figure 2.5 and Figure 2.6 show the convergence diagrams of the NLS errors via Monte Carlo simulations for various pool sizes during the curve fitting of the time series *Dsc*_20 and *Prm*_08 using Runge-Kutta (RK4) method in order to solve the dynamical system numerically for grid and random approaches, according to Algorithms 2 and 3. While the NLS errors for grid approach are plotted as solid curves, the NLS errors for random approach are shown as dashed curves for various pool sizes from 256 to 4096. The same color is used for each pool size, for the NLS errors of both approaches. After an oscillation at the beginning of the algorithm, we see the trend about minimization of the error in Figure 2.5 and Figure 2.6. This pattern is generally seen on the results of the other time series as well. We observe the convergence of the NLS errors via the Monte Carlo simulations. Table A.1 and Table A.2 give the details of the NLS error results for all time series in our data set.

The NLS error values that grid approaches converge are smaller than that of random approach for relatively small pool sizes in Figure 2.6 and Tables A.1-A.2. For large pool sizes, they approach each other, consistent with the law of large numbers. This is an important result suggesting that we may prefer the grid approach when we need to use less number of initial parameter vectors for the optimization problem.

Bold numerical values in each row of Tables A.1-A.6 indicate the minimum of the test results according to parameter pool sizes. Table 2.7 summarizes Table A.1 and Table A.2 to show the winners for the grid and random approaches with respect to the averaged NLS error values for Dsc and Prm time series. In comparison, we observe that tests with grid approach with the parameter pool $\mathbb{K}$ whose size is 512 is generally

Table 2.8 : Number of the average NLS winners according to parameter types of the each pool sizes.

Sequence type	Parameter type	Different parameter vector lengths 256	512	1024	2048	4096	Total
Dsc	grid	20	20	9	17	6	72
	random	0	0	11	3	14	28
Prm	grid	20	20	10	16	10	76
	random	0	0	10	4	10	24
Total	grid	40	40	19	33	16	148
	random	0	0	21	7	24	52

enough to find a feasible solution for the dynamical system. When we compare the other results there is no more significant gain to increase the size of the parameter pool $\mathbb{K}$. Moreover, grid approach is better than random approach for 32 time series out of 40 time series with respect to this criteria. Table 2.7 shows the distribution of the successful approach numbers among the time series for the pools of different sizes.

Alternatively, to examine the grid and the random approaches over the parameter pool sizes, we count the winner of the sequences which is the minimum values of the parameter type pairs for each pool size. In Table 2.8, we show the number of the average NLS winners whose more detailed test results are shown in Table A.1 and Table A.2.

2.4.2 The comparison of the two approaches according to MIF

Maximum improvement factor (MIF) is defined as the ratio of the final NLS error to the initial NLS error. Generally, the smaller MIF corresponds to a better performance. Besides the NLS error, MIF is an important performance metric on the evaluation of the overall optimization process. Using MIF, we track the success of the algorithm while trying to find a better solution under optimization constraints. While Table A.3 displays the resulting average MIF values for Dsc time series group to compare the grid and random approaches via simulation results, Table A.4 illustrates the corresponding resulting average MIF values for Prm time series group. We obtain Figure 2.7 by using Algorithm 1, and it shows better improvement results for the initial parameters generated by the grid approach than that of the random approach. Also, the oscillations

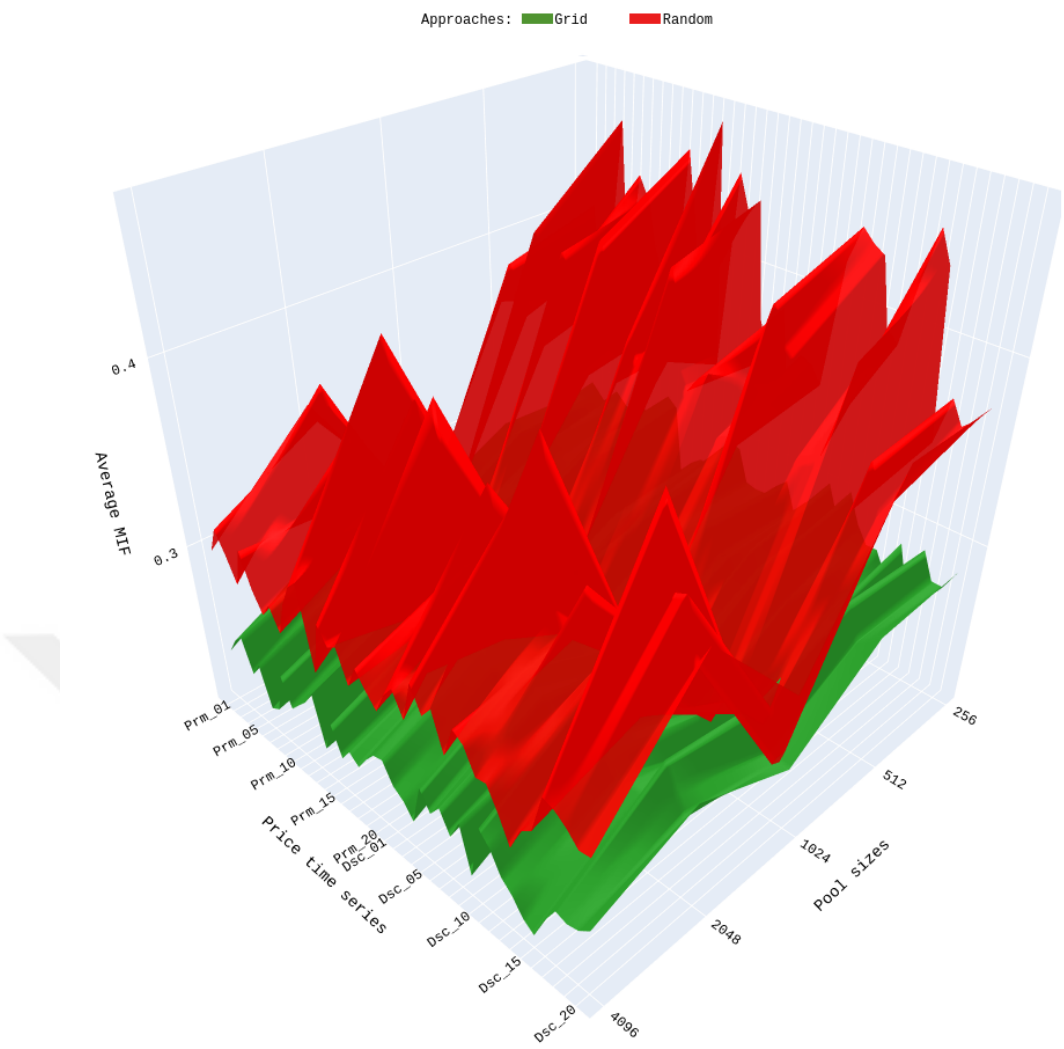

Figure 2.7 : Comparison of the grid and random approaches for all time series in the dataset according to average MIF.

with initial parameters generated by the grid approach are more stable than that of the random approach. They show that the grid approach is more reliable than the random approach for any size of the initial parameter vector pools.

Figure 2.8 and Figure 2.9 display the convergence diagrams of the MIF via Monte Carlo simulations for different pool sizes during the curve fitting of the time series *Dsc*_20 and *Prm*_08, via Algorithms 2 and 3. They show that the converged MIF values via the grid approach are better than that of the random approach.

Table 2.9 shows the MIF winners from Table A.3 and Table A.4 with respect to pairwise comparison of the two approaches for each pool size.

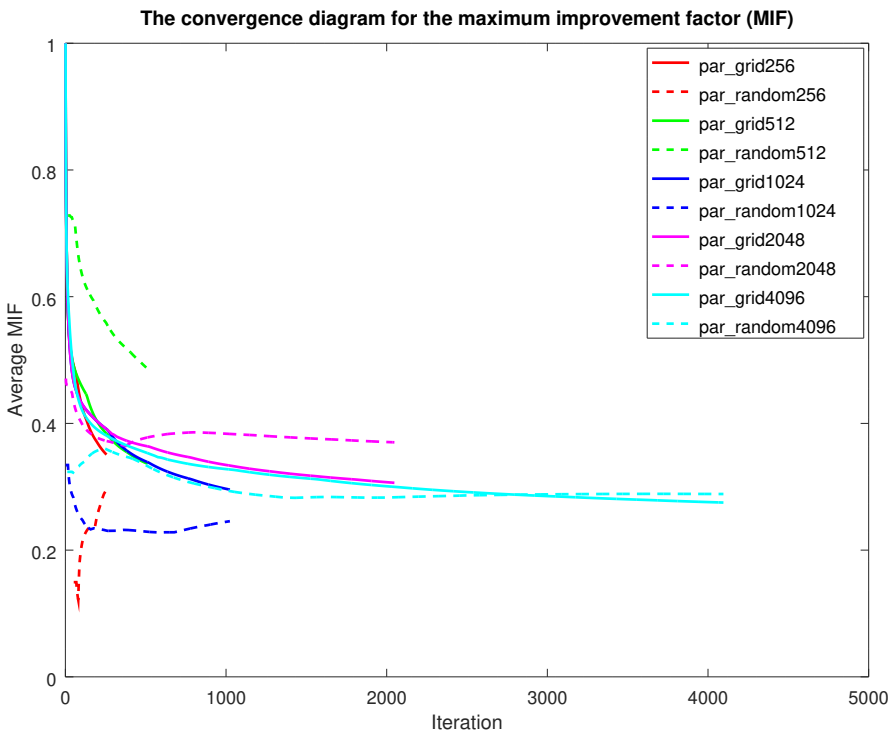

Figure 2.8 : Monte Carlo simulation of the maximum improvement factor (MIF) for curve fitting of Dsc_20 for each approach.

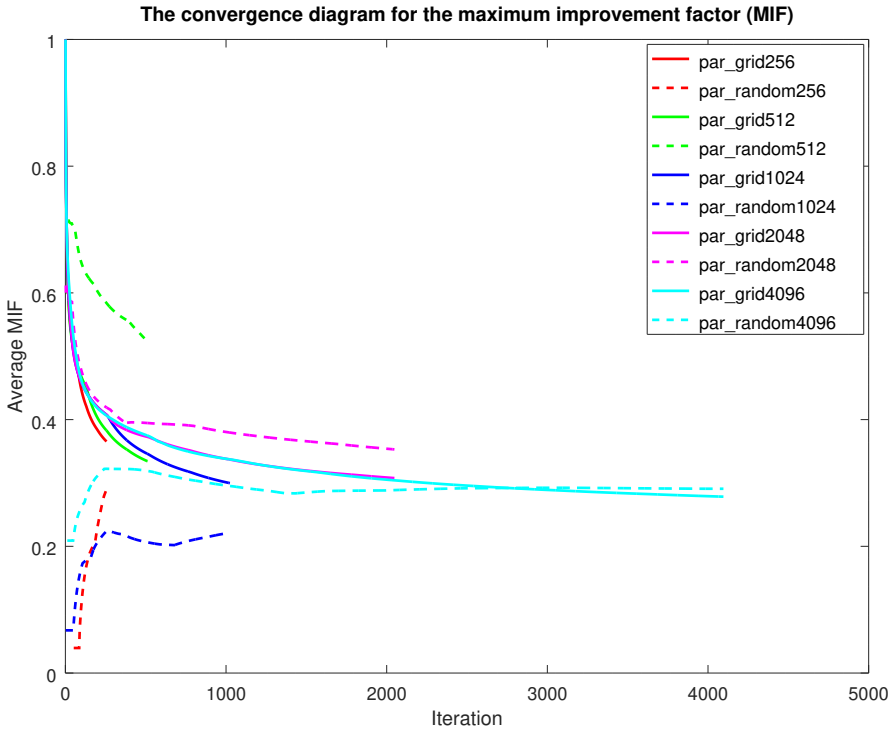

Figure 2.9 : Monte Carlo simulation of the maximum improvement factor (MIF) for curve fitting of Prm_08 for each approach.

Table 2.9 : Number of the average MIF winners according to parameter types of the each pool sizes.

Sequence type	Parameter type	Different parameter vector lengths					
		256	512	1024	2048	4096	Total
Dsc	grid	20	20	14	20	20	94
	random	0	0	6	0	0	6
Prm	grid	20	20	14	20	20	94
	random	0	0	6	0	0	6
Total	grid	40	40	28	40	40	188
	random	0	0	12	0	0	12

2.4.3 The comparison of the two approaches according to QN iteration number

While the average number of QN iterations for Dsc time series group is displayed in Table A.5 via Monte Carlo simulation results, Table A.6 shows the average number of QN iterations for Prm time series group, for the pair of grid and random approaches.

Figure 2.10 shows the dependence of the average QN iteration of the two approaches on the size of initial parameter pool for both Dsc and Prm time series according to Table A.5 and Table A.6 where results are obtained by the Algorithm 1. We observe the general rise trend of the number of QN iteration as the size of the initial parameter pool increases for both approaches. This is an expected situation. Moreover, the number of QN iteration for the grid approach is larger than that of random approach mostly. In Table A.5 and Table A.6, we can also see the result that algorithm ends with lower number of iterations for random approach in much more cases.

The winner counts of the QN iterations in Table 2.10 are calculated from Table A.5 and Table A.6 with respect to pairwise comparison.

We apply Monte Carlo simulation to the number of quasi-Newton iterations for both approaches and obtain convergence diagrams. For example, Figure 2.11 and Figure 2.12 show the convergence diagrams of the number of quasi-Newton iterations over time series *Dsc*_20 and *Prm*_08 for grid and random approaches, using Algorithms 2 and 3.

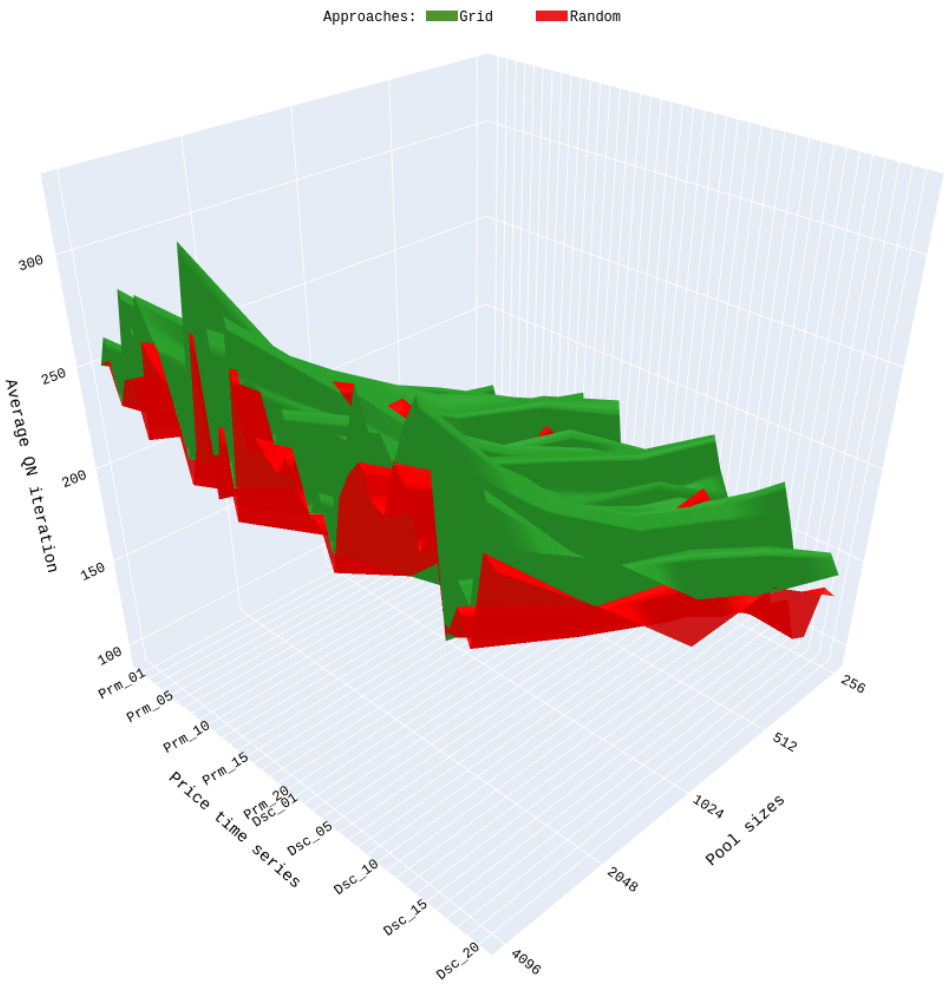

Figure 2.10 : The Comparison of the two approaches according to QN iteration number.

Table 2.10 : Number of the average QN iteration winners according to parameter types of the each pool sizes.

Sequence type	Parameter type	Different parameter vector lengths					
		256	512	1024	2048	4096	Total
Dsc	grid	4	4	5	5	2	20
	random	16	16	15	15	18	80
Prm	grid	1	4	5	5	1	16
	random	19	16	15	15	19	84
Total	grid	5	8	10	10	3	36
	random	35	32	30	30	37	164

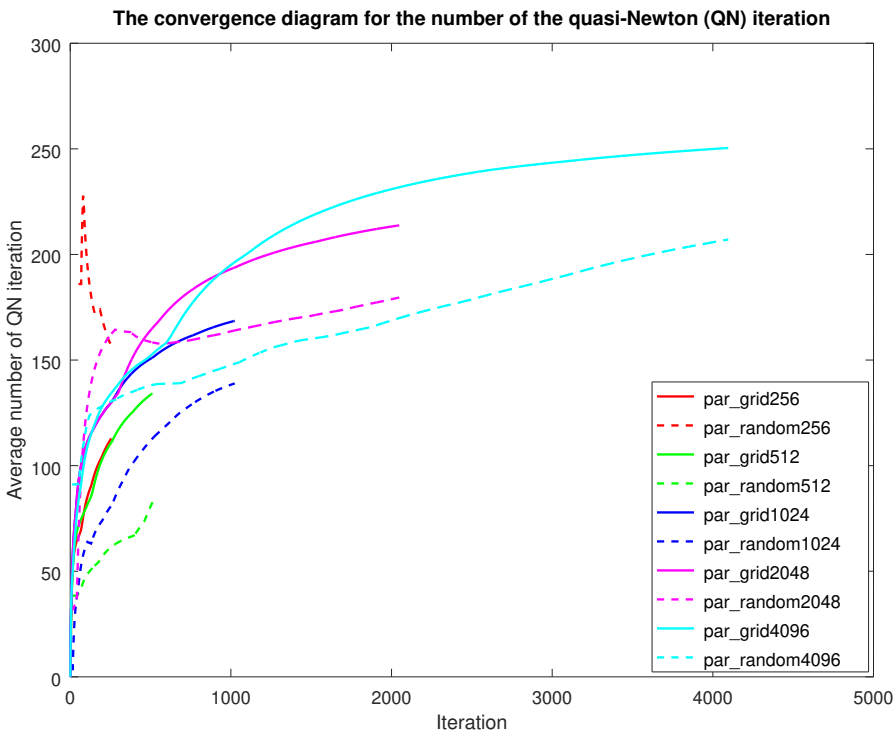

Figure 2.11 : Monte Carlo simulation of the number of quasi-Newton iteration for curve fitting of Dsc_20 for each approach.

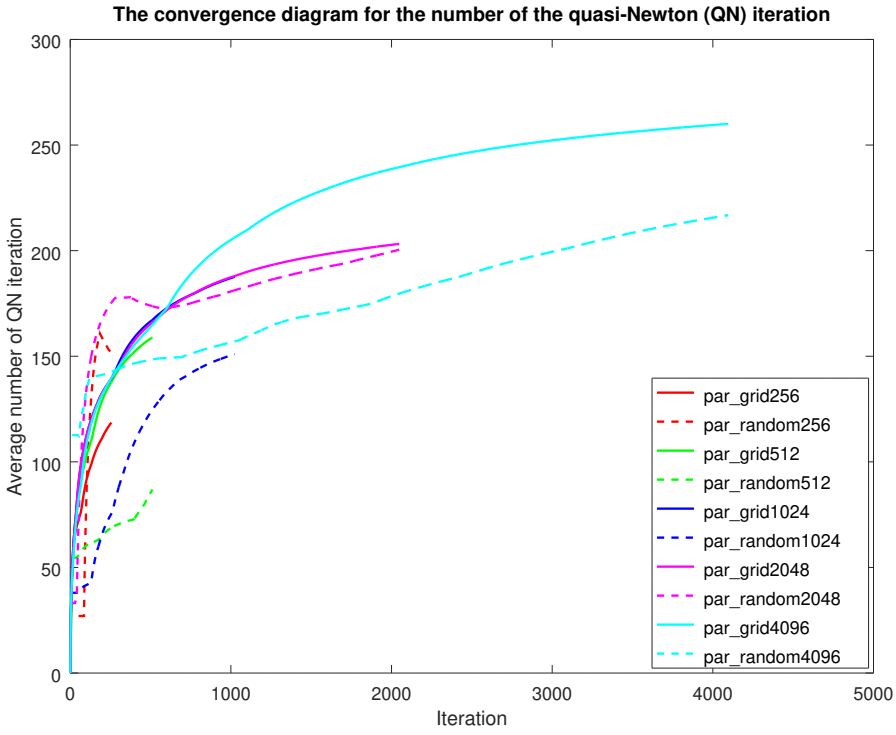

Figure 2.12 : Monte Carlo simulation of the number of quasi-Newton iteration for curve fitting of Prm_08 for each approach.

3. EVALUATION OF A NEW PARALLEL NUMERICAL PARAMETER OPTIMIZATION ALGORITHM FOR A DYNAMICAL SYSTEM

In this chapter, we study parallel optimization with initial parameter vector (IPV) pools related to nonlinear dynamical systems and present a numerical parameter optimization algorithm. A serial algorithm called the asset flow optimization forecast algorithm was prepared and an inverse problem having parameter optimization for the asset flow differential equations (AFDEs) has been used for a set of stocks in the set of closed-end funds (CEFs) traded on the NYSE (see [4]). The optimization algorithm contains a quasi-Newton (QN) weak line search [40,41] and a semi-dynamic initial parameter pool [4]. Daily market prices (MPs) and net asset values (NAVs) are used to find the parameter vectors in the AFDEs via curve fitting for the previous n days without knowing the reference functions explicitly. Runge–Kutta (RK4) method is employed to solve the dynamical system numerically and a nonlinear least squares (NLS) technique with initial value problem approach is applied based on the MP variable. The study in this chapter was published in "AIP Conference Proceedings" with title "Effectiveness of grid and random approaches for a model parameter vector optimization" after presentation at the "2nd International Conference ”Numerical Computations: Theory and Algorithms (NUMTA)" [10].

There is no algorithm that will warranty the number of required iterations to obtain the region of the global optimum (see [25], Chapter 23). In order to deal with this challenging problem in different financial market situations, we need adequately large number of IPVs generated by suitable methods and incorporated in the optimization process via high performance computing using Message Passing Interface (MPI) parallel programming [44]. It may take several days to run the sequential code in order to obtain optimal parameters with large number of IPVs to be used for stock price forecasting. When the parallel programming is used, the total time to obtain a high quality parameter vector will be reduced and this may be useful for a trader using

daily closing prices. Moreover, it is important to measure the role of large number of IPVs on the success of the optimization.

We use MPI parallel programming and analyze the success of the optimization process depending on the number of IPVs for a new parallel hybrid algorithm to estimate the model parameter vectors. Duran and Tuncel [11] tested for 64, 128, 256 and 512 cores on the Ege Server (see [45], HP ProLiant BL2x220c G5 Blade) using the 512 IPVs. They obtained speed-up for the simulated MP and NAV time series of length 1000 to run up to 512 cores. Unlike the project report [11], we deal with more extensive financial market situations and analyze the convergence of the model parameter vector, the NLS error and maximum improvement factor (MIF) to measure the success of the optimization process depending on the number of IPVs and the number of CPU cores. Moreover, we examined the behavior of the time series of length 500 and 2000. We achieved speed-up to run up to 512 cores.

The remainder of this chapter is organized as follows: First, we use the parallel nonlinear parameter optimization algorithm with classified IPV pools described in the project report [11], with new design of experiments. We use the 3rd version of AFDEs and the related problem constraints (see [7] and [8]) in this chapter. Then the convergence results of the numerical parameter optimization depending on the number of IPVs and the role of volatility are discussed.

3.1 Convergence Results of The Parameter Optimization Depending on The Number of IPVs and The Role of Volatility

We produce time series pairs as proxy to MP and NAV by using random walk simulation where the volatilities of the time series are similar to that of real CEFs traded on NYSE (see [21] and [32]). Table 3.1 displays the design and threshold values for the numerical optimization process. Table 3.2 explains the simulated MP and NAV time series of length 500, 1000, and 2000 with their volatility behavior in terms of standard deviation, price ranges and status of stock MP at a discount/premium to its NAV where P and D stand for premium and discount, respectively. The parameters in Tables 3.1-3.2 are chosen by considering the problem constraints, time constraints, available computing resources, and financial feasibility to reflect various financial

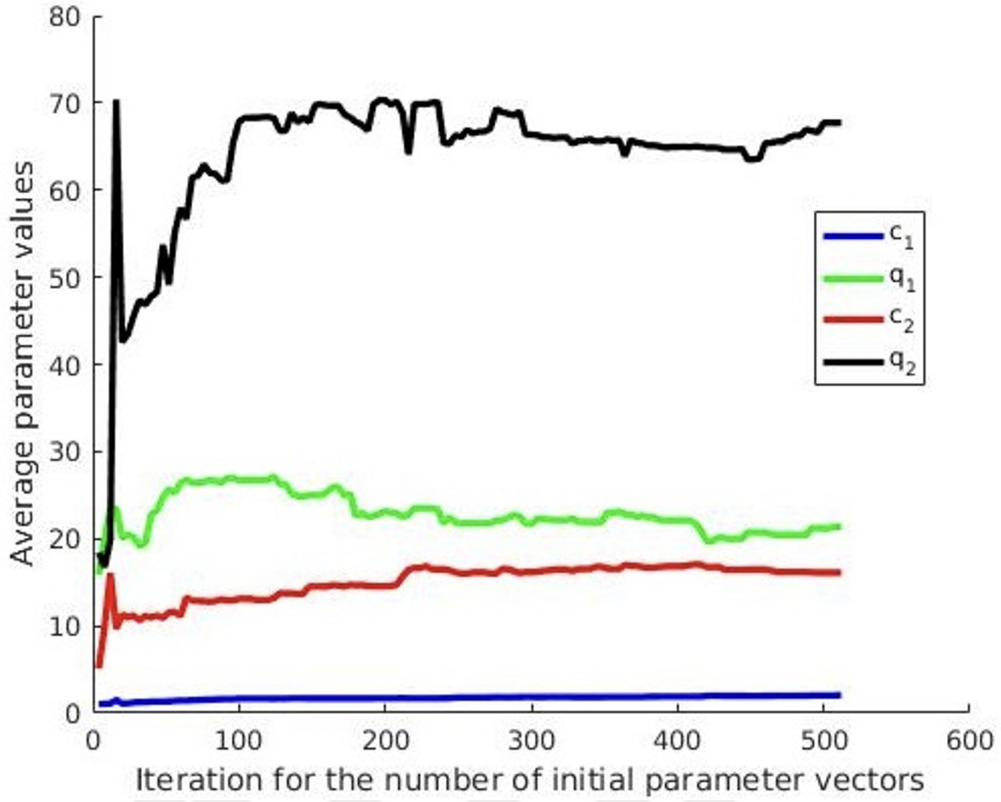

Figure 3.1 : The convergence diagram of the model parameters for the curve fitting via Monte Carlo simulation using 1k_v8 as the number of IPVs increases up to 512.

market situations generating different curves having behaviors such as almost steady, uptrend, downtrend, strong uptrend and strong downtrend in the design of experiment. The problem constraints are discussed in [4,7,21] and [8].

Table 3.2 illustrates the Monte Carlo simulation results for the parameter vector, the average NLS error and the average MIF defined as the ratio of the final NLS error to the initial NLS error. Generally, the smaller MIF corresponds to a better performance, which depends on the proximity of the IPV to the optimal one as well. Figure 3.1, Figure 3.2 and Table 3.2 show that the computed optimal parameter values, the average NLS errors, and the average MIF can converge to certain values within corresponding small ranges smoothly, after fluctuations.

We compare the serial algorithm with fixed initial parameter pool having 64 IPVs and the parallel algorithm having 512 IPVs in the classified pool and we obtain smaller NLS errors in Figure 3.3 and better MIF in Figure 3.4 via the parallel algorithm for the price time series 1k_v6, 1k_v7 and 1k_v8. The better performance in terms of errors

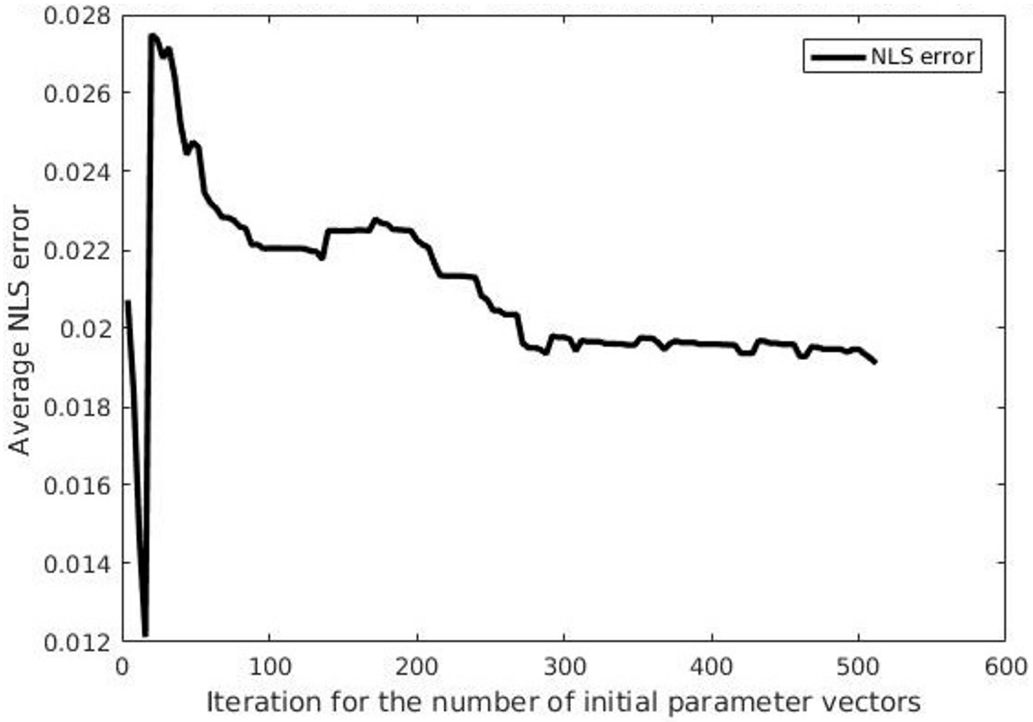

Figure 3.2 : The convergence diagram of the NLS error for the curve fitting using 1k_v8 by Monte Carlo simulation as the number of IPVs increases up to 512.

of the parallel algorithm compared to the serial one can be explained by the usefulness of larger number of IPVs.

Table 3.1 : The computational optimization by finding parameter vector in the AFDE for a large sample data set. QN method with weak line search is applied.

	Event period	RK4 method step size	# of parameter vectors in pool	Threshold for the gradient	Threshold for the NLS error
Dataset 1	5	0.05	56	10-5	0.16
Dataset 2	5	0.05	64	10-5	0.16
Dataset 3	5	0.05	≤ 512	10-5	0.16

Moreover, the average NLS error of the time series having relatively high volatility is higher than that of the time series having low volatility in Figure 3.5. For example, 0.5k_v1 - 0.5k_v4 versus 0.5k_v5 - 0.5k_v8. In general, the NLS error is larger for the time series of length 1000 as well when the volatility is sufficiently larger for both MP and NAV.

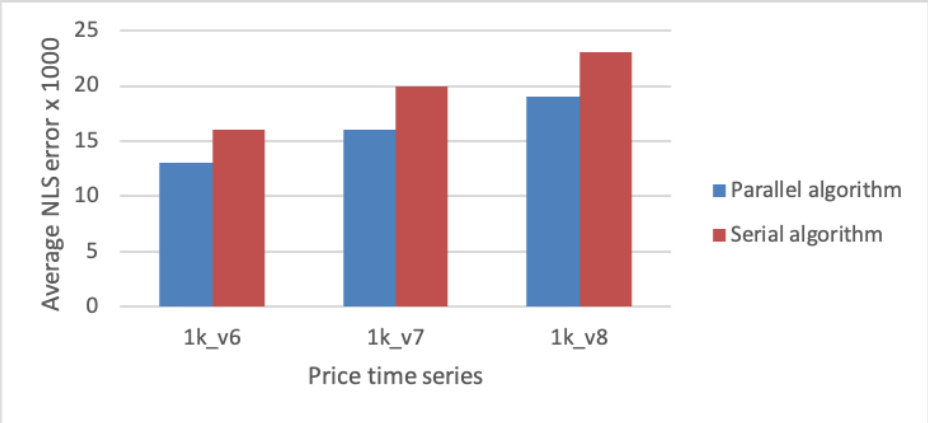

Figure 3.3 : The performance comparison of the serial algorithm with fixed initial parameter pool having 64 IPVs versus the parallel algorithm having 512 IPVs in the classified pool, in terms of NLS errors, in Table 3.2.

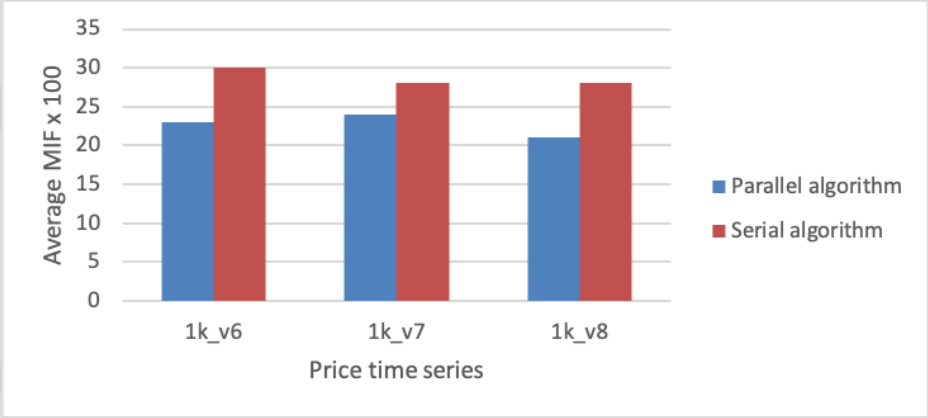

Figure 3.4 : The comparison of the serial algorithm with fixed initial parameter pool having 64 IPVs versus the parallel algorithm having 512 IPVs in the classified pool, in terms of MIF, in Table 3.2.

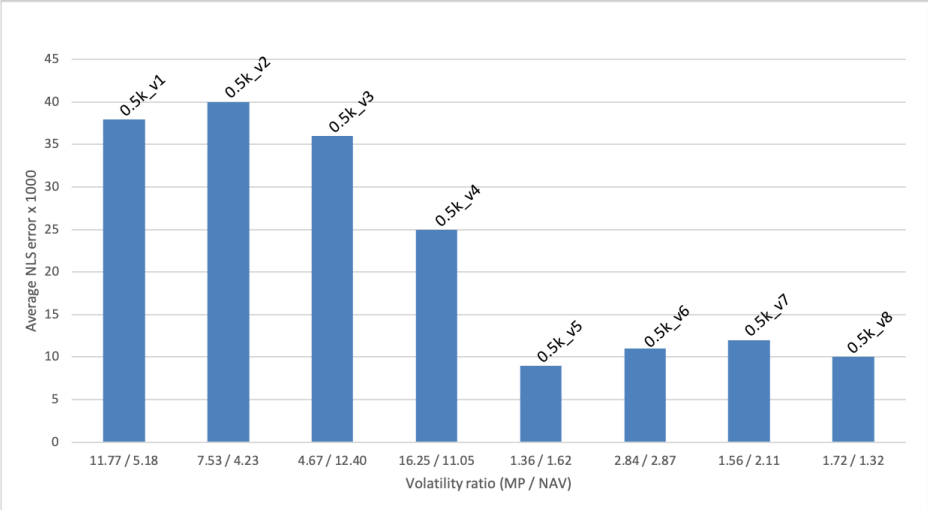

Figure 3.5 : The average NLS error comparison for the time series having various volatility levels.

Table 3.2 : Description of the time series and Monte Carlo simulation results for various number of IPVs.

Price time series (PTS)	Size of PTS	# of IVP	Volatility ratio (MP / NAV)	MP range [Min - Max]	NAV range [Min - Max]	Status (Initial / Final)	Parameter vector				Average NLS Error	Average MIF
							c_1	q_1	c_2	q_2		
0.5k_v1	500	56	11.77 / 5.18	53.60 - 99.00	45.57 - 67.23	P / P	0.97	25.89	42552	41.38	0.038	0.11
0.5k_v2	500	56	7.53 / 4.23	52.88 - 81.22	54.45 - 78.47	P / P	0.87	44.65	23468	138.29	0.040	0.18
0.5k_v3	500	56	4.67 / 12.40	50.09 - 69.70	43.10 - 80.76	P / D	0.99	33.05	46631	90.14	0.036	0.16
0.5k_v4	500	56	16.25 / 11.05	39.29 - 87.65	38.90 - 83.11	P / P	35796	13.82	16.30	44.01	0.025	0.12
0.5k_v5	500	56	1.36 / 1.62	48.58 - 54.24	50.51 - 57.25	D / D	18629	18.70	45210	47.65	0.009	0.23
0.5k_v6	500	56	2.84 / 2.87	46.98 - 58.10	45.52 - 55.89	D / P	25569	14.93	45245	30.57	0.011	0.25
0.5k_v7	500	56	1.56 / 2.11	49.11 - 56.89	47.58 - 55.79	D / P	20821	21.51	16.93	44.50	0.012	0.28
0.5k_v8	500	56	1.72 / 1.32	48.75 - 56.11	47.71 - 55.24	D / D	36161	23651	16.55	45287	0.010	0.26
1k_v1	1000	512	3.94 / 2.25	48.19 - 65.68	48.32 - 58.03	P / P	34700	18.80	18.79	45.44	0.012	0.20
1k_v2	1000	512	5.01 / 2.38	48.77 - 67.50	53.68 - 63.94	P / D	45171	21.59	16.94	57.74	0.016	0.21
1k_v3	1000	512	2.66 / 1.81	49.38 - 61.68	49.99 - 58.78	P / P	45079	16.43	17.84	40.51	0.012	0.20
1k_v4	1000	512	2.04 / 1.51	46.69 - 57.45	50.58 - 57.89	P / D	15707	17.97	13.60	34.97	0.015	0.24
1k_v5	1000	512	7.16 / 4.75	53.76 - 78.61	51.42 - 71.40	P / P	35796	20.79	16.46	55.83	0.017	0.22
1k_v6	1000	64	3.63 / 3.15	42.00 - 56.92	47.92 - 60.18	P / D	12785	15.27	14.57	38.42	0.016	0.30
1k_v6	1000	512					45232	18.13	18.00	45.96	0.013	0.23
1k_v7	1000	64	3.63 / 2.91	51.82 - 67.31	48.45 - 61.55	P / P	17533	25.97	44997	51.89	0.020	0.28
1k_v7	1000	512					21186	22.87	16.68	52.33	0.016	0.24
1k_v8	1000	64	7.87 / 2.37	54.49 - 85.44	47.97 - 57.84	P / P	15707	26.76	13.26	56.68	0.023	0.28
1k_v8	1000	512					45018	21.41	45246	67.72	0.019	0.21
2k_v1	2000	512	5.64 / 4.02	50.05 - 71.68	52.80 - 71.20	P / D	27030	16.61	19.44	36.87	0.013	0.23

4. EVALUATING THE MATURITY OF OPENFOAM SIMULATIONS ON GPGPU FOR BIO-FLUID APPLICATIONS

We investigated the challenges facing CFD solvers as applied to bio-medical fluid flow simulations and in particular the OpenFOAM 2.1.1 solver, icoFoam, for the large penta-diagonal matrices coming from the simulation of blood flow in arteries with a structured mesh domain in PRACE-3IP project at TGCC Curie (a modern Tier-0 system) (see [46] and references therein). We generated a structured mesh by using blockMesh as a mesh generator tool. To decompose the generated mesh, we employed the decomposePar tool. After the decomposition, we used icoFoam as a flow simulator/solver tool. We achieved scaled speed-up for large matrices up to 64 million x 64 million matrices and speed-up up to 16384 cores on Curie thin nodes. The study in this chapter was published in "Proceedings of the Emerging Technology (EMiT) Conference" with title "Evaluating the maturity of OpenFOAM simulations on GPGPU for bio-fluid applications" [12].

In this chapter, we examined OpenFOAM 2.2.2 "icoFoam" simulator with an iterative solver such as diagonal incomplete LU preconditioned bi-conjugate gradient in addition to direct solvers such as distributed SuperLU 4.0 (see [13]). The flow problem produced various matrices as the time advances in simulation. The solution of the matrices obtained after each time step can be more challenging due to the changing structure of the matrices. This change may be caused by mess change or flow variable change. Generally the solution time of the matrices increases as the time advances in simulation.

It is challenging to discuss on the benefits or drawbacks of hybrid nodes. There are tradeoffs using GPU accelerators especially for the software packages or applications where it is not possible to fit the whole part into GPU. While it is expected to obtain a reduced time due to the accelerator, there would be communication over-head between the various processors and the GPU accelerators, as well. Therefore, it

is important to obtain a feasible/optimal proportion of the tasks to MPI, OpenMP, and CUDA/OpenCL usages in emerging CPU+GPU systems. For example, it is not possible to do everything only in GPU for a complex algorithm like SuperLU_DIST. Therefore hybrid nodes like Curie hybrid nodes at CEA in France provide opportunity.

It would be interesting to discuss about the relative energy requirements for thin nodes versus hybrid nodes. A diversification of hardware solutions based on the application capability may be needed in order to attain a good efficiency (see [47] and [48]). While the compute partition of Curie thin nodes having total of 80,640 cores consumes 2132 kW, the partition of Curie hybrid nodes having total of 288 Intel + 288 Nvidia processors uses 108.80 kW as the total power (see TOP500 Supercomputing sites [49] and the Green500 List [50]). The partition of Curie hybrid nodes outperforms the Curie thin nodes when the energy efficiency is compared in terms of performance per watt and the rates of computation are 1,010.11 MFLOPS/W and 637.43 MFLOPS/W, respectively.

The remainder of this chapter is organised as follows: In Section 4.1, the test environment and the flow of approach are described. In Section 4.2, thin nodes results of the CPU performance for the iterative solver icoFoam and the hybrid parallel codes (MPI+OpenMP) of a direct solver SuperLU_DIST 4.0 are compared. Moreover, simulation test results of hybrid node using MPI+OpenMP+CUDA versus MPI+OpenMP with SuperLU_DIST 4.0 solvers are presented.

4.1 Test Environment and Flow of Approach

OpenFOAM (see [51]) is an open source Computational Fluid Dynamics (CFD) toolbox. It is a software package with many tools for several main tasks of the simulation such as pre-processing for meshing, decomposition and solution. Here, the solver refers to not only linear system solver but also Navier Stokes solver and simulator.

The first four matrices in Table 4.1 are obtained at time 0.00005 (s) of the simulation where the time step size is 0.00005 (s), as in [46]. Unlike [46], the last six matrices in Table 4.1 are encountered at the third time step, at time 0.012 (s) of the simulation

Table 4.1 : Description of matrices.

	N	NNZ	NNZ/N	Origin
mC_8M	8,000,000	39,988,000	4.999	ITU Mathematics
mC_16M	16,000,000	79,984,000	4.999	ITU Mathematics
mC_6M_D	6,000,000	41,800,000	6.967	ITU Mathematics
mC_8M_D	8,000,000	55,760,000	6.970	ITU Mathematics
mC_8M_n	8,000,000	39,988,000	4.999	ITU Mathematics
mC_16M_n	16,000,000	79,984,000	4.999	ITU Mathematics
mC_20M_n	20,000,000	99,982,000	4.999	ITU Mathematics
mC_6M_n_D	6,000,000	41,780,000	6.963	ITU Mathematics
mC_8M_n_D	8,000,000	55,760,000	6.970	ITU Mathematics
mC_10M_n_D	10,000,000	69,660,000	6.966	ITU Mathematics

where the time step size is 0.004 (s). This is a relatively large time step size for such a very small mesh size. Thus, we obtained challenging ill-conditioned matrices. Almost 5 or 7 banded sparse matrix occurs at each time step and the matrices are described in Table 4.1. The flowchart in Figure 4.1 shows the flow of approach in the paper.

4.2 Test Results

The tests were done for only a few time steps due to time limitations, while the real case runs are conducted for more than thousands of time steps. No single CPU solution was possible because of long waiting times, so, information regarding the pre-processing (meshing), partitioning etc. are given for parallel processing. The most time consuming part of the simulation was the decomposing of the mesh. For 8192 partitions, it took over 3 hours. The "Simple" decomposition method was preferred since the running cases were for a structured mesh. This technique simply splits the geometry into pieces by direction, such as 32 pieces in x direction and 32 pieces in y direction. Since the mesh is structured, mC_20M_n matrix means 20M of cells in the fluid domain.

4.2.1 Thin node results

We compared the CPU performance of the iterative solver icoFoam and the hybrid parallel codes (MPI+OpenMP) of a direct solver SuperLU_DIST 4.0 (see [13]) at TGCC Curie (a Tier-0 system) thin nodes at CEA, France (see [14]). Figure 4.2 and Figure 4.3 show the wall-clock time comparisons of the solvers, excluding the

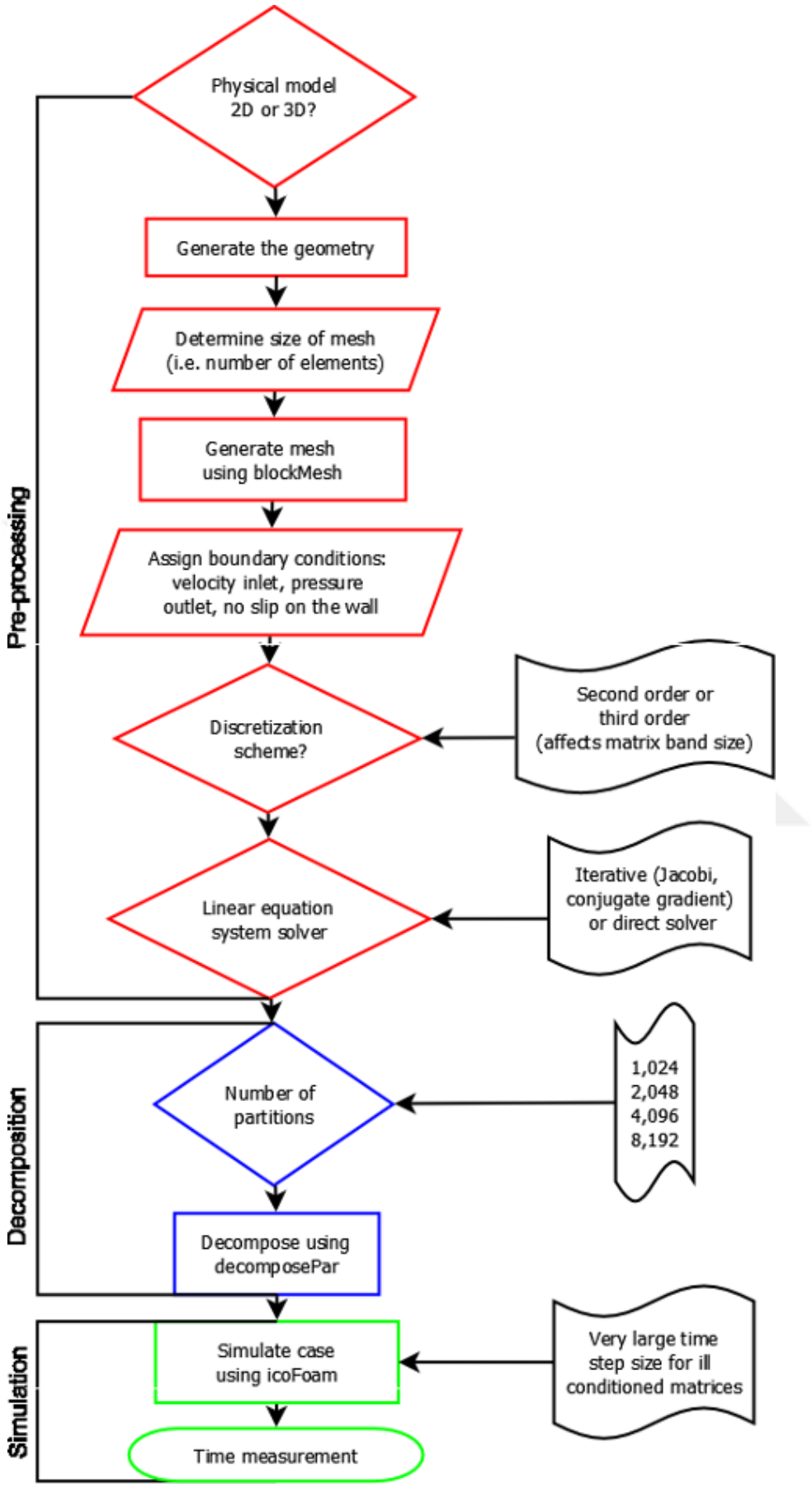

Figure 4.1 : Flowchart for the flow of the approach including the main tasks.

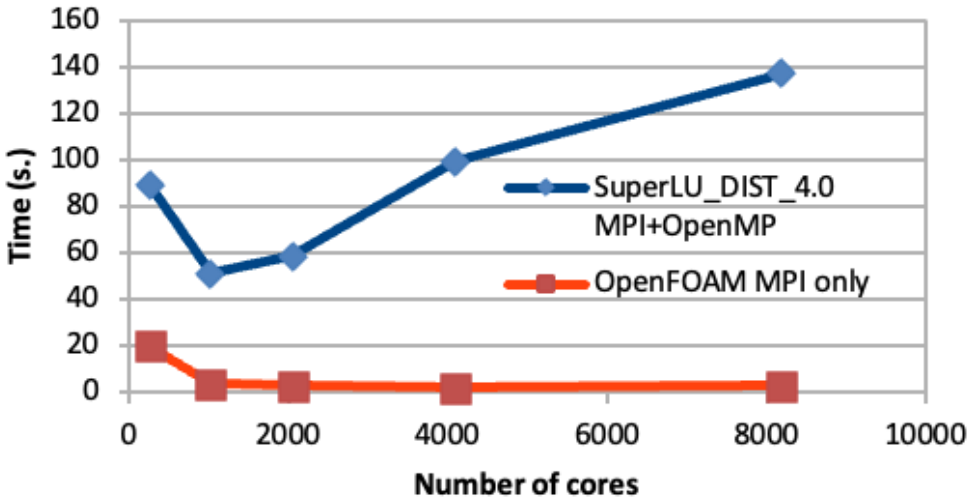

Figure 4.2 : Wall-clock time comparison of the solvers for mC_16M_n on Curie thin nodes.

refinement time, for mC_16M_n and mC_20M_n on Curie thin nodes, respectively. The iterative solver with a diagonal incomplete LU preconditioned bi-conjugate gradient outperforms the direct solver SuperLU_DIST 4.0 for the simulation matrices.

4.2.2 Hybrid node results using MPI+OpenMP+CUDA

We compared the performance of the hybrid parallel codes of MPI+OpenMP+CUDA (see [52]) versus MPI+OpenMP implementation of SuperLU_DIST 4.0 at TGCC Curie (a Tier-0 system) hybrid nodes of CPU + GPU at CEA, France (see [14]). Table 4.2 describes the corresponding configurations while we run the direct solver.

Table 4.2 : The Configuration of MPI+OpenMP and MPI+OpenMP+CUDA for the direct solver.

Testbed:CURIE/	hybrid	hybrid	hybrid	hybrid
SuperLU_DIST version	4	4	4	4
# of cores	64	256	512	1024
# of processes	16	64	128	256
# of threads per process	4	4	4	4
# of GPUs per process	1	1	1	1

Table 4.3 shows the performance results for the ten simulation matrices described in Table 4.1. For example, Figure 4.4 shows the comparison for the performances of MPI+OpenMP+CUDA and MPI+OpenMP implementations for mC_20M_n on Curie

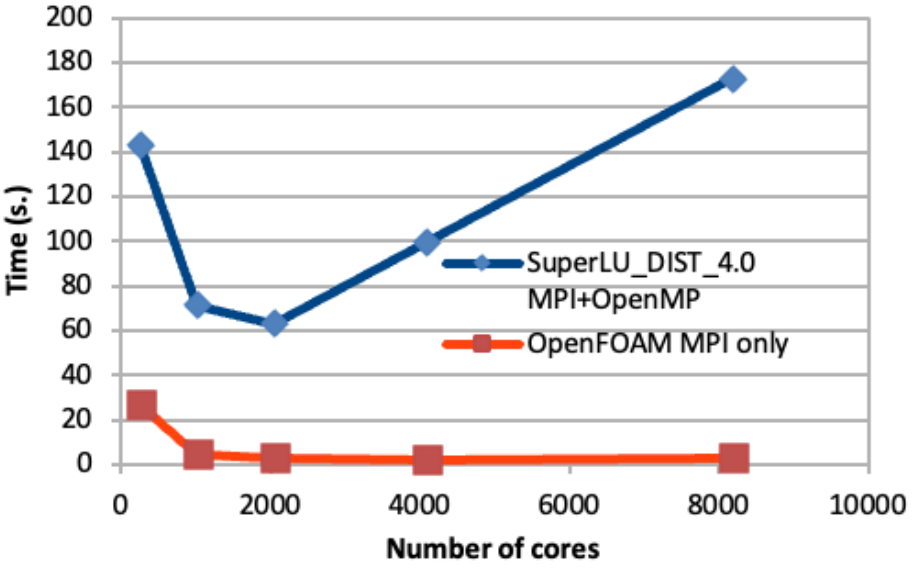

Figure 4.3 : Wall-clock time comparison of the solvers for mC_20M_n on Curie thin nodes.

hybrid nodes. In Figure 4.5, we observe a linear speed-up of the direct solver up to 512 cores for both implementations for mC_20M_n on Curie hybrid nodes.

Generally, we see that MPI+OpenMP implementation outperforms the hybrid of MPI+OpenMP+CUDA for this set of simulation matrices when we consider the wall clock times for the optimal number of cores because of several overheads coming from CUDA implementation for the direct solver algorithm. It is not possible to put everything only in GPU for SuperLU_DIST. Therefore, the tasks should be proportioned to MPI, OpenMP, and CUDA/OpenCL. In SuperLU_DIST 4.0 (see [52]), cuBLAS library execution is one of the most time consuming tasks performed in GPU in order to gain from explicit parallelization. On the other hand, there are overheads such as data transfer on PCIe between host and device memory (CPU and GPU) and new data structure changes related to data packing and scattering. Moreover, SuperLU is a complex algorithm and it is challenging to select the right combination for better intra-node communications and inter-node communications within CPU+GPU heterogeneous systems, under current technology limitations (see [53]).

The last eight matrices in Table 4.3 are challenging large matrices because they are relatively denser or ill-conditioned. The error labelled Error 1 occurs for small number

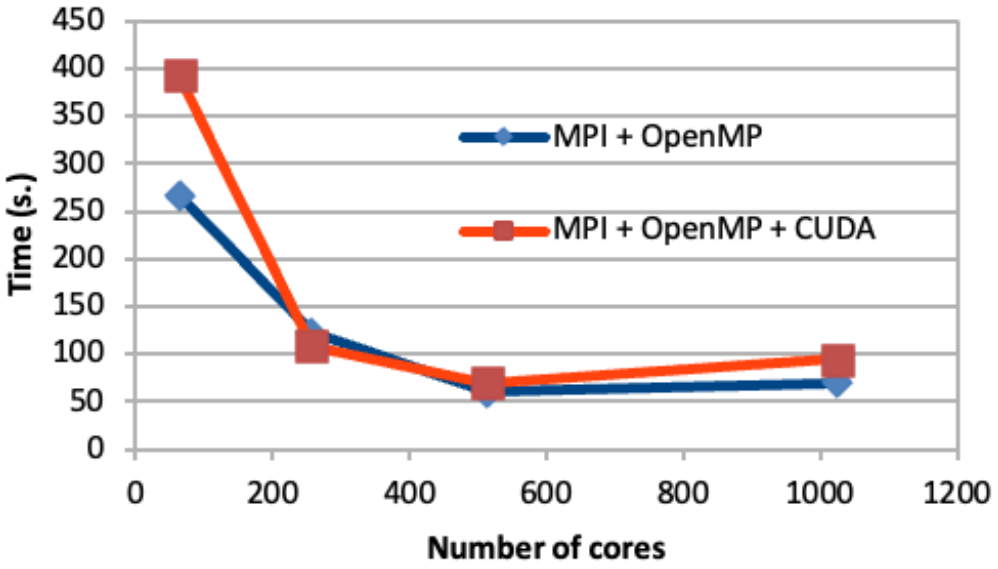

Figure 4.4 : Wall-clock time of direct solver for mC_20M_n on Curie hybrid nodes.

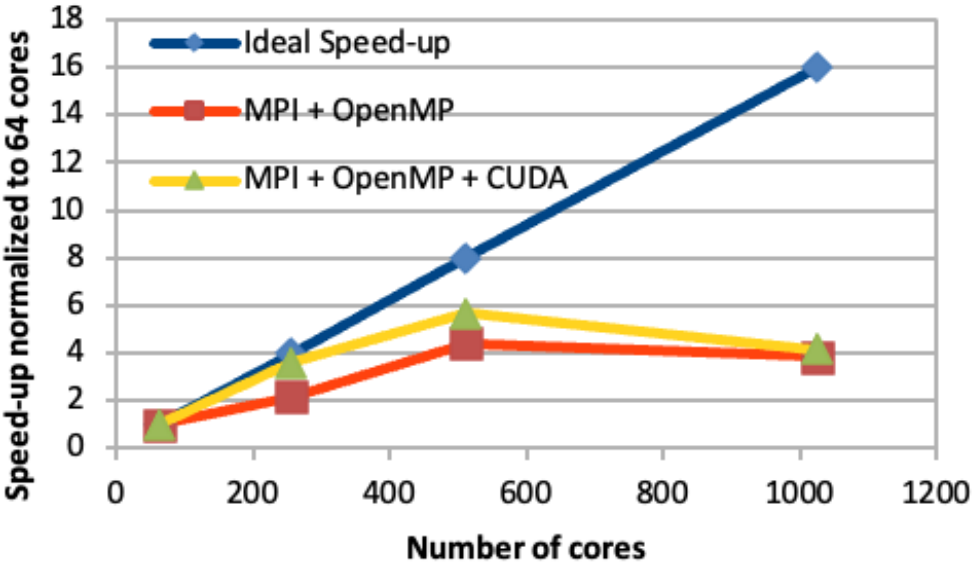

Figure 4.5 : Speed-up of direct solver for mC_20M_n on Curie hybrid nodes.

Table 4.3 : Wall Clock Times (s) of SuperLU_DIST 4.0 for the large penta-diagonal matrices for 2D problems and hepta-diagonal matrices for 3D problems, described in Table 4.1, on MPI+OpenMP versus MPI+OpenMP+CUDA implementations.

Matrices / Number of cores		64	256	512	1024
mC_8M	MPI + OpenMP	99.96	34.70	28.78	37.89
	MPI + OpenMP + CUDA	94.70	39.10	43.70	60.72
mC_16M	MPI + OpenMP	230.30	83.19	47.73	59.02
	MPI + OpenMP + CUDA	236.83	87.23	60.00	81.41
mC_6M_D	MPI + OpenMP	Error 1	260.38	296.74	239.52
	MPI + OpenMP + CUDA	Error 1	Error 2	254.44	257.15
mC_8M_D	MPI + OpenMP	Error 1	1005.96	516.86	387.20
	MPI + OpenMP + CUDA	Error 1	680.25	Error 2	353.40
mC_8M_n	MPI + OpenMP	94.70	31.00	32.79	35.83
	MPI + OpenMP + CUDA	70.94	38.27	Error 3	61.34
mC_16M_n	MPI + OpenMP	181.53	75.93	49.53	58.61
	MPI + OpenMP + CUDA	233.22	75.58	61.42	83.61
mC_20M_n	MPI + OpenMP	266.82	122.59	60.30	69.49
	MPI + OpenMP + CUDA	393.49	108.90	69.60	94.99
mC_6M_n_D	MPI + OpenMP	1178.51	409.15	248.84	211.70
	MPI + OpenMP + CUDA	782.22	294.14	Error 2	222.04
mC_8M_n_D	MPI + OpenMP	Error 1	948.03	533.78	386.72
	MPI + OpenMP + CUDA	Error 1	682.02	Error 2	349.16
mC_10M_n_D	MPI + OpenMP	Error 1	877.92	465.60	373.09
	MPI + OpenMP + CUDA	Error 1	752.78	Error 2	Error 3

of cores. We meet with an error message labelled Error 2 related to buffer size during the factorization subroutine pdgstrf, for the hepta-diagonal matrices. Error 3 is a CUDA stream error related to setting cuBLAS library execution stream.

5. SPECTRAL EFFECTS OF LARGE MATRICES FROM OIL RESERVOIR SIMULATORS ON PERFORMANCE OF SCALABLE DIRECT SOLVERS

We design a novel hybrid algorithm and solver for large sparse linear systems. First, we consider scalable direct solvers because of their robustness and examine the SuperLU_DIST 3.3 (see Li et al. [13]) for distributed memory parallel machines among several sparse direct solvers (see Li et al. [13], Li and Demmel [54], Amestoy et al. [55], Schenk and Gartner [56,57], Duran and Saunders [58], Duran et al. [59] and references contained therein). Duran et al. [60] discussed the advantages and limitations of the SuperLU solvers and tested the code of SuperLU_DIST 3.0 (see [13]) in order to measure the performance scalability for various sparse matrices (see [61] for the theoretical foundation regarding the distribution of eigenvalues for some sets of random matrices). SuperLU_DIST needs to be improved for certain types of challenging sparse matrices. The study in this chapter was published, and presented at the "SPE Large Scale Computing and Big Data Challenges in Reservoir Simulation Conference" with title "Spectral effects of large matrices from oil reservoir simulators on performance of scalable direct solvers" [15].

We believe that the approach for exception handling of challenging matrices via Gerschgorin circles is beneficial and practical to stabilize the performance of the solvers. Nearly defective matrices are among the challenging matrices. Clustered eigenvalues observed via Gerschgorin circles may be used to detect nearly defective matrix.

The presence of repeated eigenvalues can be one of the sources of challenges. The repeated eigenvalue may have fewer eigenvectors than the multiplicity of eigenvalue. While such eigenvalue is called defective eigenvalue, the corresponding matrix is referred as a defective matrix (see [62]). If the matrix of eigenvectors is singular, then the matrix cannot be diagonalizable and the matrix is defective. We observe that it takes longer time to solve sparse linear system having defective or nearly defective

matrix than regular matrix. Moreover, defective matrix may lead to memory restriction due to the appearance of more fill-ins than that of diagonalizable matrix.

The remainder of this chapter is organized as follows. First, the test matrices are described. Later, the computation for spectral properties is presented and several illustrative examples are given.

5.1 Methods and Results

The selected eigenvalues of large matrices are computed using the Scalable Library for Eigenvalue Problem Computations (SLEPc) software (see [63]), which is developed based on the Portable, Extensible Toolkit for Scientific Computation (PETSc) (see [64]). The code has been tested up for all sparse matrices in the list on HP Integrity Superdome SD32B (see [65]), a computing server with shared memory architecture at UHeM (see [66]). The software package includes implementations of a set of methods for the solution of large sparse eigenproblems on parallel computers. It is applicable to both symmetric and nonsymmetric matrices. In our computations, we used the Krylov-Schur method available in the package.

We can compute all eigenvalues of the small randomly populated matrices and show the distribution of eigenvalues for RAND_30K_75 in Figure 5.1. We observe that nearly all eigenvalues can be found within the circle except for the largest eigenvalue that is indicated by an isolated point in figure. The distribution of eigenvalues for a randomly populated matrix is a good reference for other patterned matrices in order to understand the deviations between them (see [67]). We describe the test matrices in Table 5.1.

Table 5.1 : Description of matrices.

Matrices	Order	NNZ	NNZN	Origin	Kind of problem
RAND_30K_75	30000	2250000	75	UHeM	Randomly populated
Matrix300k	900000	13362067	14,85	Reservoir simulation	Black-oil model
spe5Ref_dpdp_a	2058000	66808700	32,46	Reservoir simulation	7 component EOS model
spe5Ref_dpdp_b	2058000	71260352	34,62	Reservoir simulation	7 component EOS model
spe5Ref_dpdp_c	2058000	68930222	33,49	Reservoir simulation	7 component EOS model
spe5Ref_dpdp_d	2058000	68930222	33,49	Reservoir simulation	7 component EOS model
spe5Ref_dpdp_e	2058000	67189220	32,65	Reservoir simulation	7 component EOS model
EMILIA_923	923136	40373538	43,74	UFSMC	Geomechanical structural
HELM2D03LOWER_20K	392257	1939353	4,94	UHeM	Patched matrix obtained from HELM2D03
M_UHEM3	1425825	17037638	11,94	UHeM	Patched matrix obtained from parabolic_fem
mC_8M	8000000	39988000	4,999	UHeM	CFD

For the large sparse matrices we compute the extreme eigenvalues. We try to see a rough picture of the distribution for the rest of the eigenvalues by using Gerschgorin's theorem. We show the Gerschgorin's circles of the patched matrix M_UHEM3 (see Duran et al. [68]), five matrices from 7 component EOS model, matrix Emilia_923, and matrix HELM2D03LOWER_20K in Figures 5.2 - 5.9, respectively. As the matrix becomes more patterned, the spectral space changes and the eigenvalues take place within disjoint, overlapped or clustered of Gerschgorin circles.

For example, when we examine the spectral properties of HELM2D03LOWER_20K, the real parts of the eigenvalues range between 2.294563 and 4.944602 with many repeated eigenvalues. Those clustered eigenvalues can be observed via Gerschgorin circles. Therefore, HELM2D03LOWER_20K is a nearly defective matrix. We used the SuperLU_DIST 3.3 with tunings of super-nodal storage parameters. However, it runs slowly for the matrix HELM2D03LOWER_20K compared to EMILIA_923, because HELM2D03LOWER_20K is a challenging matrix. It takes approximately 7,5 times longer than EMILIA_923, although HELM2D03LOWER_20K's order, total number of non-zeros and the number non-zeros per row are less than that of EMILIA_923.

SuperLU_MCDT is a distributed direct solver and the software will be uploaded to website (see [17]) after academic permissions from Istanbul Technical University. Here, we used symbolic factorization, ParMETIS (see [69]) for column permutation and Intel MKL (see [70]) as the BLAS library, among several options. The tuning of super-nodal storage parameters is important for the performance and we selected the tuned parameters relax:100 and maxsuper:110 (see [64]).

We define an optimal minimum number of cores as the number of cores that provides the minimum wall clock time for a given size of problem, where a right match occurs between the problem size and the available resources such as memory, in presence of communication overhead (see Duran et. al [71]). We find that the optimal minimum number of cores required depends on the sparsity level and size of the matrix. As the sparsity level of matrix decreases and the order of matrix increases, we expect that the optimal minimum number of cores increases slightly.

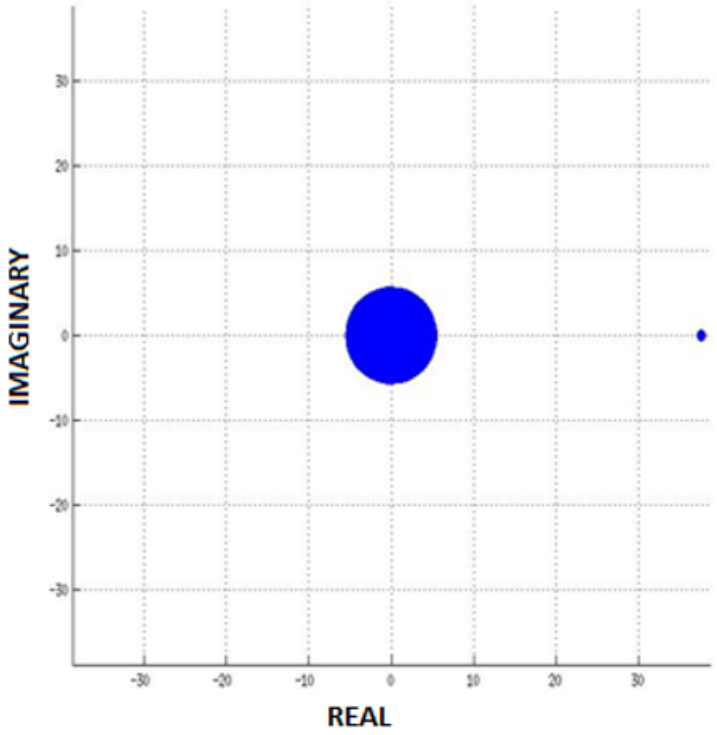

Figure 5.1 : Distribution of eigenvalues for matrix RAND_30K_75.

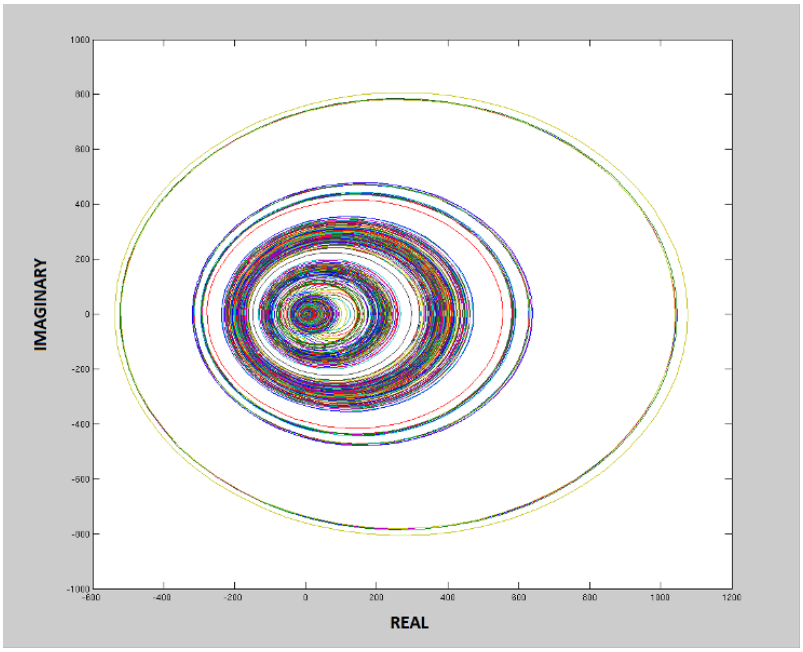

Figure 5.2 : Gerschgorin's circles of M_UHEM3.

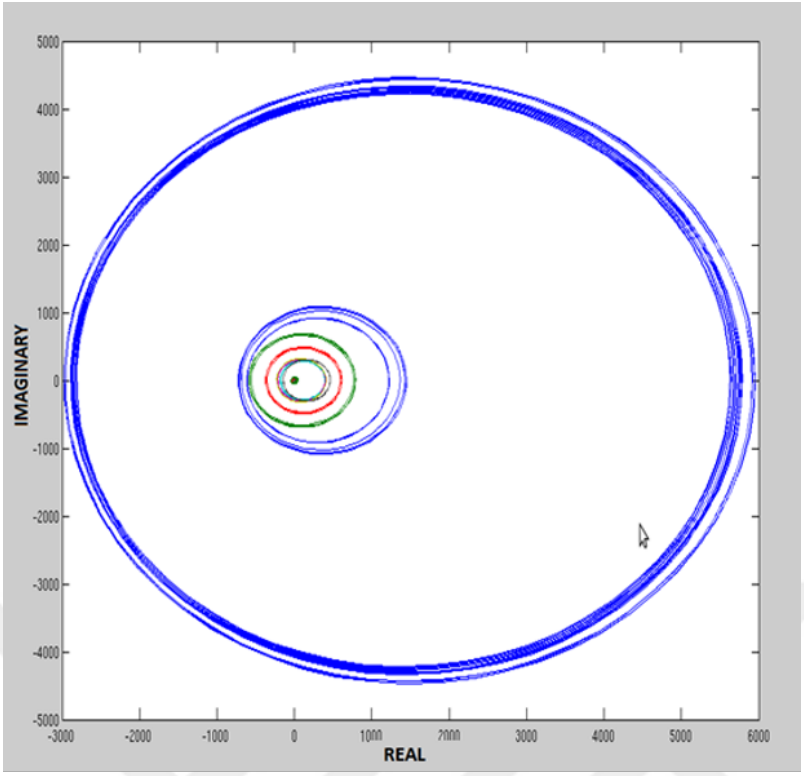

Figure 5.3 : Gerschgorin's circles of spe5Ref_dpdp_a.

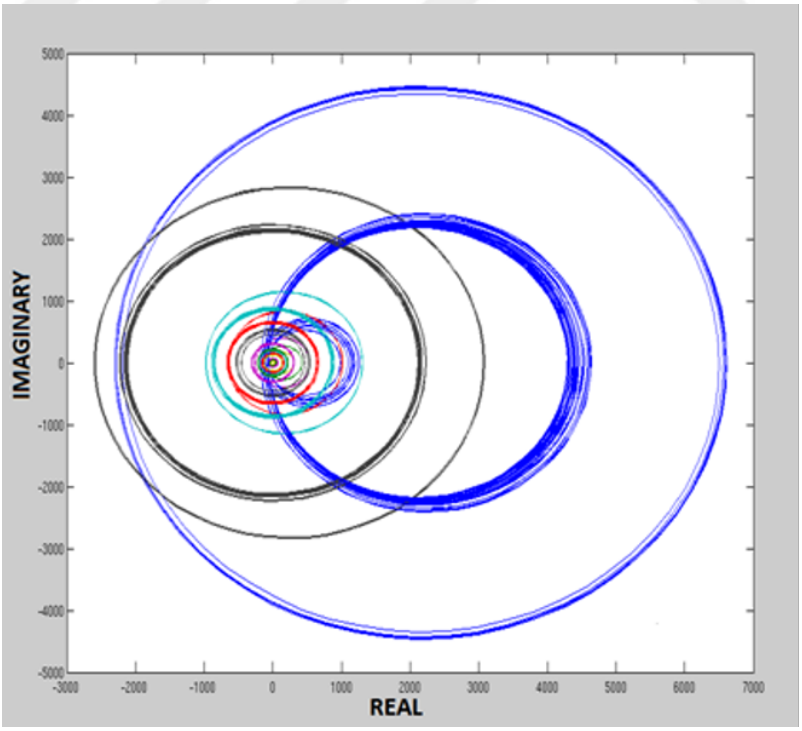

Figure 5.4 : Gerschgorin's circles of spe5Ref_dpdp_b.

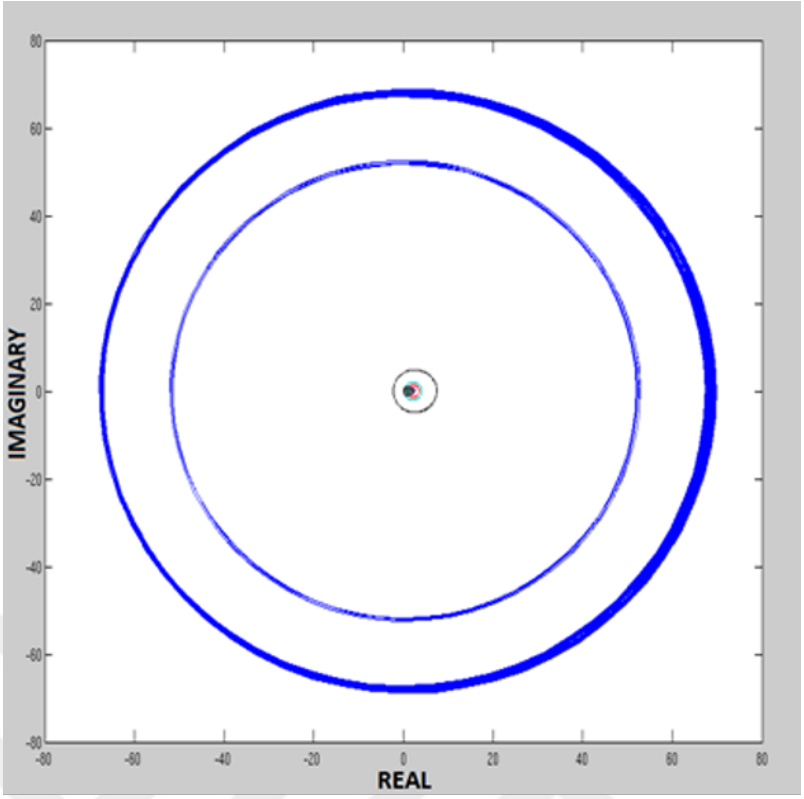

Figure 5.5 : Gerschgorin's circles of spe5Ref_dpdp_c.

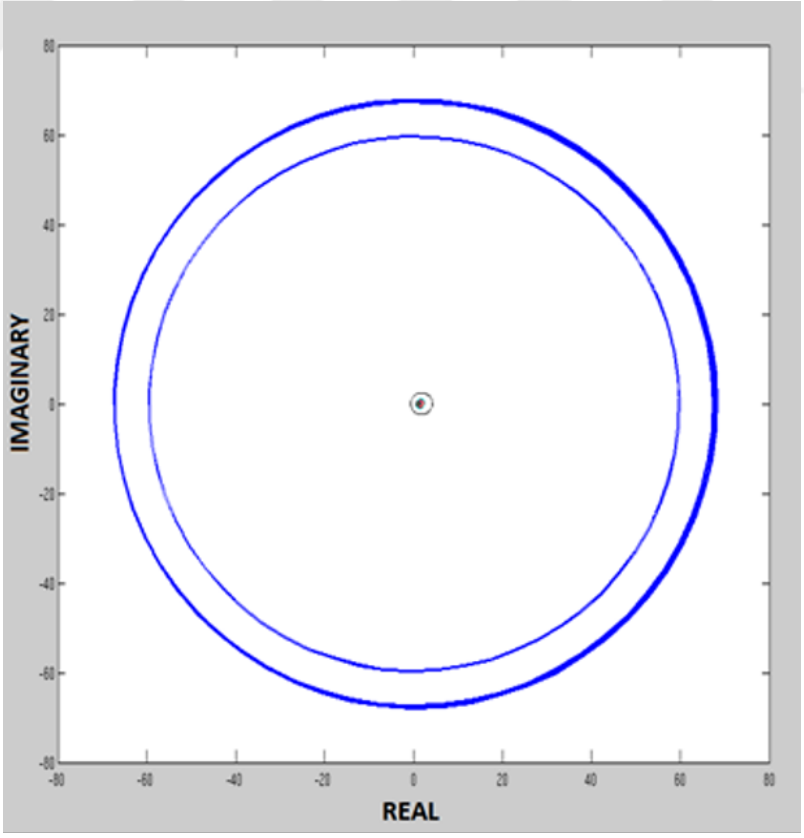

Figure 5.6 : Gerschgorin's circles of spe5Ref_dpdp_d.

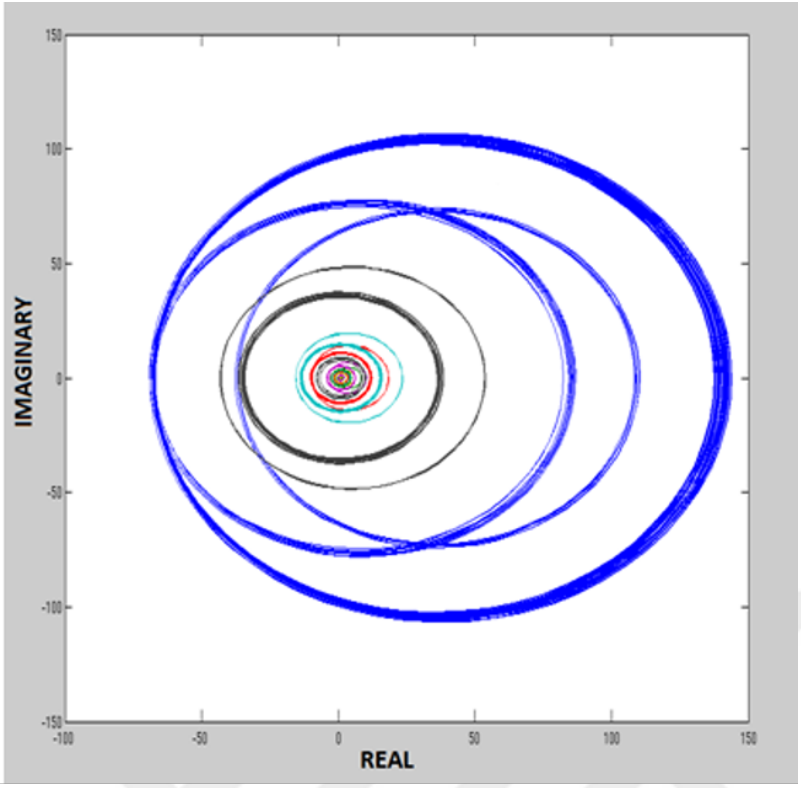

Figure 5.7 : Gerschgorin's circles of spe5Ref_dpdp_e.

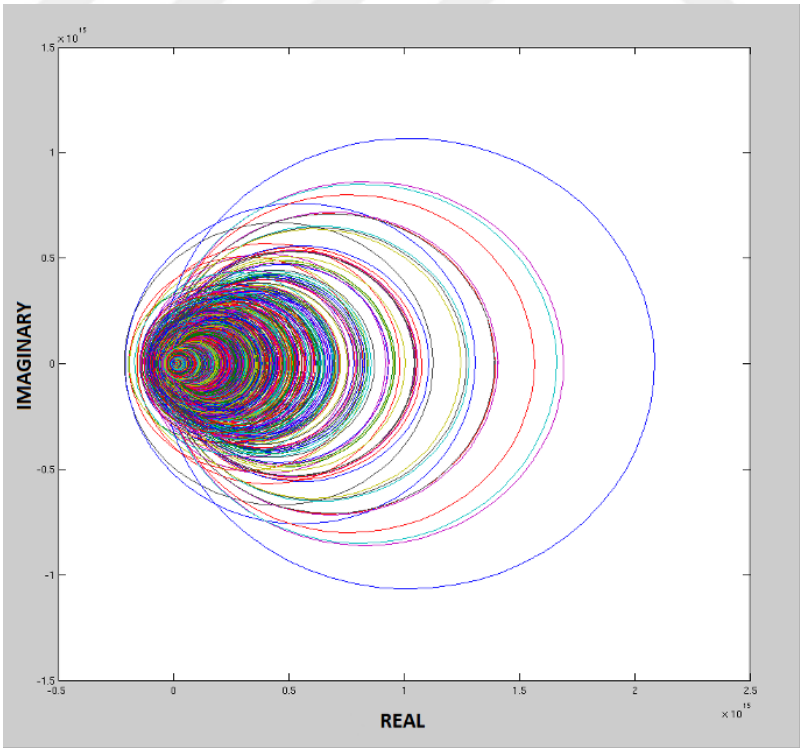

Figure 5.8 : Gerschgorin's circles of matrix Emilia_923.

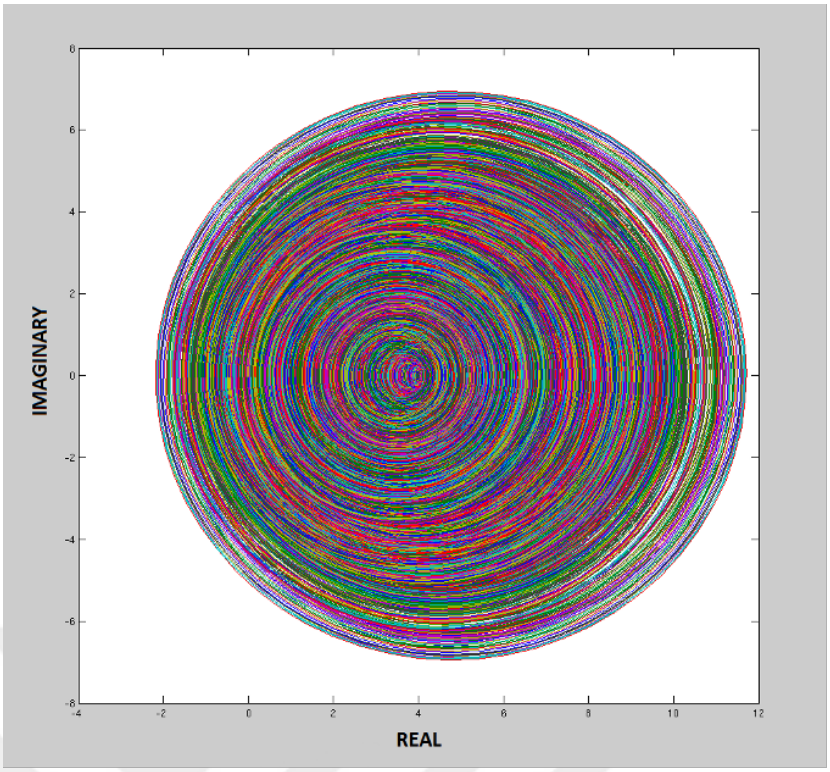

Figure 5.9 : Gerschgorin's circles of matrix HELM2D03LOWER_20K.

Table 5.2 : Optimal wall clock times (s) of SuperLU_MCDT for the Matrix300k from the black-oil model and five matrices from 7 component EOS model described in Table 5.1.

Matrices		Optimal time (s)	Optimal minimum number of cores (meshes)
Matrix300k	Factor Time	10,46	1024
	Total Time	24,48	(256x4)
spe5Ref_dpdp_a	Factor Time	52,79	16384
	Total Time	208,27	(4096x4)
spe5Ref_dpdp_b	Factor Time	49,29	16384
	Total Time	220,91	(4096x4)
spe5Ref_dpdp_c	Factor Time	193,6	1024
	Total Time	242,34	(256x4)
spe5Ref_dpdp_d	Factor Time	193,54	1024
	Total Time	242,11	(256x4)
spe5Ref_dpdp_e	Factor Time	51,43	16384
	Total Time	216,49	(4096x4)

Table 5.2 illustrates the time for the factorization and the total time for each matrix based on the optimal minimum number of cores. We observe that the optimal minimum number of cores can be different depending on the matrix properties.

We imbedded direct solvers (kernel class) such as SuperLU_DIST 3.3 and SuperLU_MCDT in addition to the solvers provided by OpenFOAM (see [23]). Because future exascale systems are expected to have heterogeneous and many-core distributed nodes, we believe that our SuperLU_MCDT software is a good candidate for future systems. We tested the performance of the solver at TGCC Curie (a Tier-0 system) at CEA, France (see [14] and [71]). SuperLU_MCDT worked up to 16384 cores for the large penta-diagonal matrices for 2D problems and hepta-diagonal matrices for 3D problems, arising from the incompressible blood flow simulation, without any problem. For example, Table 5.3 shows the distribution of wall clock time (s) for mC_8M matrix and the impact of number of super-nodes and the communication overhead coming from ParMETIS on the performance. We obtained similar results for the other matrices in Table 5.1. SuperLU_MCDT uses dense block structures, called super-nodes to get advantages of BLAS3 (see [72]) with the common technique of array padding, like SuperLU_DIST 3.3. Super-node detection differs as process mesh size and its square or rectangular shape. So we observe sometimes more efficient case matched to the super-node detection strategies of the algorithm where the optimal minimum number of cores for the matrix mC_8M is 512.

Table 5.3 : Distribution of wall clock time (s) for mC_8M matrix using ParMETIS for column permutation, at TGCC Curie (a Tier-0 system) at CEA, France

# of cores (mesh)	256 (16 X 16)	512 (16 X 32)	1024 (32 X 32)	2048 (32 X 64)	4096 (64 X 64)	8192 (64 X 128)	16384 (128 X 128)
Nonzeros in L	736867161	80858737	759889256	765376719	692260216	700475156	690287571
Nonzeros in U	736867161	80858737	759889256	765376719	692260216	700475156	690287571
nonzeros in L+U	1465734322	160717474	1511778512	1522753438	1376520432	1392950312	1372575142
nonzeros in LSUB	102386047	11558966	106262844	108045660	94662608	97338383	96491385
# of super-nodes	204238	26847	207025	208620	215465	214535	217216
Equil time	0,39	0,27	0,53	1,41	2,07	2,23	6,05
RowPerm time	2,18	0,27	2,17	2,18	2,18	2,2	2,17
ColPerm time	5,54	8,63	31,12	66,29	102,04	139,54	301,12
SymbFact time	3,92	0,41	4,07	4,1	3,57	3,66	3,63
Distribute time	1,07	0,24	0,75	0,76	0,69	0,92	1,68
Factor time	9,34	1,79	13,64	13,87	25,33	43,46	90,98
Solve time	3,33	0,01	1,59	1,88	1,59	1,85	2,05
Refinement time	19,76	1,06	7,84	6,59	7,75	8,1	10,85
||X-Xtrue||/||X||	1,18E-12	4,06E-11	1,80E-12	2,35E-12	1,12E-12	1,08E-12	1,10E-12
Total time (s)	45,53	12,68	61,71	97,08	145,22	201,96	418,53

6. CONCLUSIONS

In this thesis, we discuss many aspects of parameter optimization in mathematical modeling. We present methods and suggestions for parameter optimization in the differential equation system and for optimal selection of the computational resources and their combinations as crucial part of mathematical modeling in real-world use cases.

First of all, we propose a new mathematical method, a hybrid of hyperbox and multi-start methods, for an inverse problem of parameter vector estimation in order to understand the population dynamics of investors and obtain valuable results for machine learning applications. We examine the effect of two different approaches for obtaining initial parameter vector pools via Monte Carlo simulations during the parameter vector optimization in the dynamical system of asset flow differential equations. The Monte Carlo simulations for NLS errors, MIF values and the number of QN iterations work due to the Law of Large Numbers.

The initial parameter vectors in the initial parameter vector pool $\mathbb{K}_g$ of the grid approach are relatively more uniformly distributed than the initial parameter vectors in the $\mathbb{K}_r$ of random approach, in the hyperbox search space. We find that the performance of the grid approach is relatively better than that of random approach for selection of initial parameter vectors in a hyperbox based on our Monte Carlo simulation and convergence diagrams for NLS errors and MIF values in the unconstrained optimization problem. While the NLS error values that grid approaches converge are smaller than that of random approach for relatively small pool sizes, they approach each other for large pool sizes. This result is consistent with the Law of Large Numbers.

In sum, there is a tradeoff between the accuracy via less NLS error and the computational cost via number of QN iterations. While the grid approach outperforms the random approach in terms of NLS errors and MIF values, the random approach

requires less number of QN iterations than the grid approach during our comparisons based on the experiments for our data set.

Asset flow differential equations reflect expert opinions coming from microeconomic principals and experimental economics. Consequently, we obtain optimal / feasible parameter vector that reflects investor preferences based on current market situations.

Our approach in Chapter 2 such as grid approach and random approach in hyperbox for selection of initial parameter vectors may be applied to other appropriate optimization problems in science and engineering as well.

In Chapter 3, we complement the project report [11] about the development and assessment of the parallel nonlinear parameter optimization algorithm with classified IPV pools. In this work, we evaluated the convergence of the model parameter vector, the NLS error and MIF to quantify the success of the optimization process depending on the number of IPVs and financial market situations such as the presence of low volatility, high volatility and stock MP at a discount/premium to its NAV. We obtained smaller NLS errors and better MIF via the parallel algorithm compared to the serial algorithm with fixed initial parameter pool having less number of IPVs, based on the dataset. Moreover, we observe that generally the NLS error is larger for the time series pair as proxy to MP and NAV whose volatilities are sufficiently higher for both MP and NAV when the other variables are fixed. Finally, we consider different work scheduling and load balancing strategies. We try dynamic IPV assignments to cores. For example, first, each core can launch with one parameter vector and seek to take new one when it completes the task.

In Chapter 4, we perform bio-medical fluid flow simulations for the large matrices arising from the simulation of blood flow in arteries in emerging CPU+GPU systems. The flow problem generated various challenging matrices during the simulation. We compared the CPU performance of the iterative solver icoFoam and the hybrid parallel codes (MPI+OpenMP) of a direct solver SuperLU_DIST 4.0 (see [13]) at TGCC Curie (a Tier-0 system) thin nodes at CEA, France (see [14]). We observe that the iterative solver with a diagonal incomplete LU preconditioned bi-conjugate gradient outperforms the direct solver SuperLU_DIST

4.0 for the simulation matrices. Moreover, we compared the performance of the hybrid parallel codes of MPI+OpenMP+CUDA versus MPI+OpenMP implementation of SuperLU_DIST 4.0 at TGCC Curie (a Tier-0 system) hybrid nodes of CPU + GPU at CEA, France (see [14]). Generally, we notice that MPI+OpenMP implementation outperforms the hybrid of MPI+OpenMP+CUDA for the set of simulation matrices when we consider the wall clock times for the optimal number of cores because of several overheads coming from CUDA implementation for the complex direct solver algorithm. Furthermore, we met with several errors for the challenging simulation matrices. We believe that the technology developments in emerging CPU+GPU systems will increase the scalability of related complex algorithms by eliminating the bottlenecks coming from communication and right matching of system components required for special applications.

In Chapter 5, we observe that the existing versions of SuperLU are sensitive to challenging matrices and need exception handling. Apart from the solver, spectral analysis can be done and tuned parameters may be used accordingly. We released the first SuperLU_MCDT (Many Core Distributed) version (1.0) with several novelties based on the direct solver SuperLU_DIST 3.3. Our benchmark tests show that SuperLU_ MCDT can run on up to 16348 cores.

There is no unique solver that fits all our needs for every matrix because of the rich pattern spectrum of matrices and the NP-complete problem of best reordering for minimum fill-in. We observe that the optimal minimum number of cores can be different depending on the matrix properties. The existence of optimal minimum number of cores requires a rule base to make a decision.

We believe that expert systems (see [73]), knowledge-based computer programs with a set of inference rules ('if then' type statements) in a rule base, are among the most promising subfields in artificial intelligence for big data discovery and decision making applications such as oil and gas reservoir simulators in a timely and reliable fashion. We plan that expert system tools for real time decision making based on the spectral properties and the super-node detection strategies of various large patterned matrices coming from reservoir modeling and the exception handling for the challenging

matrices will be among the new properties of SuperLU_MCDT version (2.0). We will use an expert system with forward chaining as a reasoning method to reach conclusions in our learning algorithm.

REFERENCES

[1] **Bremermann, H.** (1970). A method of unconstrained global optimization, *Mathematical Biosciences*, *9*, 1–15.

[2] **Milstein, J.**, (1981). The inverse problem: Estimation of kinetic parameters, Modelling of Chemical Reaction Systems, Proceedings of an Int. Workshop Heidelberg Series, K. H. Ebert, P. Deuflhard, and W. J*ä*ger, (Eds.), Springer Series in Chemical Physics, volume*18*, Springer, pp.92–101.

[3] **Chapra, S.** (2008). *Applied Numerical Methods with MATLAB for Engineers and Scientists, 2nd Edition*, McGraw Hill, New York, NY.

[4] **Duran, A. and Caginalp, G.** (2008). Parameter optimization for differential equations in asset price forecasting, *Optimization Methods & Software*, *23*(4), 551–574, issue: Mathematical programming in data mining and machine learning.

[5] **Tunçel, M. and Duran, A.** (2023). Effectiveness of grid and random approaches for a model parameter vector optimization, *Journal of Computational Science*, *67*, 101960.

[6] **Caginalp, G. and Ermentrout, G.** (1991). Numerical studies of differential equations related to theoretical financial markets, *Applied Mathematics Letters*, *4*(1), 35–38.

[7] **Caginalp, G. and Balenovich, D.** (1999). Asset flow and momentum: deterministic and stochastic equations, *Philosophical Transactions of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences*, *357*(1758), 2119–2133.

[8] **Duran, A.** (2011). Stability analysis of asset flow differential equations, *Applied Mathematics Letters*, *24*(4), 471–477.

[9] **Caginalp, C.**, **Caginalp, G. and Swigon, D.** (2021). Stochastic asset flow equations: Interdependence of trend and volatility, *Physica A: Statistical Mechanics and its Applications*, *574*, 125985.

[10] **Duran, A. and Tuncel, M.** (2016). Evaluation of a new parallel numerical parameter optimization algorithm for a dynamical system, *Proceedings of the 2nd International Conference Numerical Computations: Theory and Algorithms (NUMTA2016) Italy, 19-25 June 2016, AIP Conference Proceedings*, volume*1776*, p.090052.

[11] **Duran, A. and Tuncel, M.** (2014). *Scalable Parallel Nonlinear Parameter Optimization Algorithm with Parameter Pools*, https://doi.org/10.5281/zenodo.825430, project final report.

[12] **Duran, A.**, **Piskin, S. and Tuncel, M.** (2016). Evaluating the maturity of OpenFOAM simulations on GPGPU for bio-fluid applications, *Proceedings of the Emerging Technology (EMiT) Conference*, pp.11–14.

[13] **Li, X.**, **Demmel, J.**, **Gilbert, J.**, **Grigori, L.**, **Shao, M. and Yamazaki, I.** (1999). 'SuperLU users' guide,"Lawrence Berkeley Nat. Lab., Berkeley, CA, **Technical Report**, USA, Tech. Rep., Sep. 1999. Update: 2011.

[14] *CEA TGCC Curie HPC cluster*, https://www-hpc.cea.fr/index-en.html, accessed Jan. 2016.

[15] **Duran, A. and Tuncel, M.** (2014). Spectral Effects of Large Matrices from Oil Reservoir Simulators on Performance of Scalable Direct Solvers, SPE Large Scale Computing and Big Data Challenges in Reservoir Simulation Conference and Exhibition, SPE-172984-MS.

[16] **Al-Shaalan, T.M.**, **Fung, L.S. and Dogru, A.H.** (2003). A scalable massively parallel dual-porosity dual-permeability simulator for fractured reservoirs with super-k permeability, *SPE annual technical conference and exhibition*, OnePetro.

[17] *University of Florida sparse matrix collection*, http://www.cise.ufl.edu/research/sparse/matrices/, accessed Jan. 2016.

[18] **Glasserman, P.** (2003). *Monte Carlo Methods in Financial Engineering*, Springer, New York, NY.

[19] **Dixon, M.**, **Halperin, I. and Bilokon, P.** (2021). *Machine Learning in Finance From Theory to Practice*, Springer, New York, NY.

[20] **Ganeshapillai, G.** (2014). *Learning connections in financial time series*, Massachusetts Institute of Technology, USA.

[21] **Duran, A.** (2009). Sensitivity analysis of asset flow differential equations and volatility comparison of two related variables, *Numerical Functional Analysis and Optimization*, *30*(1), 82–97.

[22] **Rinnooy Kan, A. and Timmer, G.** (1987). Stochastic global optimization methods part I: Clustering methods, *Mathematical Programming*, *39*(1), 27–56.

[23] **Rinnooy Kan, A. and Timmer, G.** (1987). Stochastic global optimization methods part II: Multi level methods, *Mathematical Programming*, *39*(1), 57–78.

[24] **Jones, D.R.**, **Perttunen, C.D. and Stuckman, B.E.** (1993). Lipschitzian optimization without the Lipschitz constant, *Journal of Optimization Theory and Applications*, *79*(1), 157–181.

[25] **Bartholomew-Biggs, M.** (2006). *Nonlinear Optimization with Financial Applications*, Springer Science & Business Media, New York, NY.

[26] **Caginalp, G.**, **Porter, D. and Smith, V.L.** (2000). Overreaction, momentum, liquidity, and price bubbles in laboratory and field asset markets, *Journal of Psychology and Financial Markets*, *1*(1), 24–48.

[27] **Smith, V.**, **Suchanek, G. and Williams, A.** (1988). Bubbles, crashes, and endogenous expectations in experimental spot asset markets, *Econometrica*, *56*(1), 1119–1151.

[28] **Duran, A. and Caginalp, G.** (2007). Overreaction diamonds: Precursors and aftershocks for significant price changes, *Quantitative Finance*, *7*(3), 321–342.

[29] **Duran, A.** (2006). *Overreaction behavior and optimization techniques in mathematical finance*, University of Pittsburgh, USA.

[30] **Ritter, G.X. and Urcid, G.** (2021). *Introduction to Lattice Algebra: With Applications in AI, Pattern Recognition, Image Analysis, and Biomimetic Neural Networks*, Chapman and Hall/CRC, Boca Raton, FL.

[31] **Higham, D.J. and Higham, N.J.** (2016). *MATLAB Guide*, SIAM, Philadelphia, PA.

[32] **Anderson, S. and Born, J.** (2002). *Closed-end Fund Pricing: Theories and Evidence*, Kluwer Academic Publishers, Boston, MA.

[33] **Nocedal, J. and Wright, S.** (2006). *Numerical Optimization*, Springer Science & Business Media, New York, NY.

[34] **Dennis, Jr, J.E. and Moré, J.J.** (1977). Quasi-Newton methods, motivation and theory, *SIAM review*, *19*(1), 46–89.

[35] **Dennis, J.E. and Schnabel, R.B.** (1996). *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, SIAM, Philadelphia, PA.

[36] **Floudas, C.A. and Pardalos, P.M.** (2008). *Encyclopedia of Optimization*, Springer Science & Business Media, New York, NY.

[37] **Jonsson, M.** (2006). *An Introduction to Monte Carlo Simulations*, Textbook for Numerical Methods with Financial Applications, Department of Mathematics, University of Michigan, Ann Arbor, MI.

[38] **Sauer, T.** (2012). *Numerical Analysis*, Pearson: 2nd Edition, New York, NY.

[39] **Ross, S.** (2006). *A First Course in Probability*, Pearson Prentice Hall 7th Edition, Upper Saddle River, NJ.

[40] **Broyden, C.G.** (1970). The convergence of a class of double-rank minimization algorithms 1. general considerations, *IMA Journal of Applied Mathematics*, *6*(1), 76–90.

[41] **Broyden, C.G.** (1970). The convergence of a class of double-rank minimization algorithms: 2. The new algorithm, *IMA Journal of Applied Mathematics*, *6*(3), 222–231.

[42] **Duran, A. and Bommarito, M.J.** (2011). A profitable trading and risk management strategy despite transaction costs, *Quantitative Finance*, *11*(6), 829–848.

[43] **Raschka, S.** (2015). *Python Machine Learning*, Packt Publishing Ltd, Birmingham, UK.

[44] **Gropp, W.**, **Gropp, W.D.**, **Lusk, E.**, **Skjellum, A. and Lusk, A.D.F.E.E.** (1999). *Using MPI: portable parallel programming with the message-passing interface*, volume*1*, MIT press.

[45] *National Center for High Performance Computing (UHeM), Technical Spesifications of Ege Server*, `http://en.uhem.itu.edu.tr/index.php/donanim-2`, accessed Feb. 2016.

[46] **Duran, A.**, **Celebi, M.S.**, **Piskin, S. and Tuncel, M.** (2015). Scalability of OpenFOAM for bio-medical flow simulations, *The Journal of Supercomputing*, *71*, 938–951.

[47] **Meyer, N. and Lawenda, M.** (2013). *D5.2: Best Practices for HPC Procurement and Infrastructure*, `https://doi.org/10.5281/zenodo.6572412`.

[48] **David, J.**, **Richet, J.N.**, **Boyer, E.**, **Anastopoulos, N.**, **Collet, G.**, **Colin de Verdière, G.**, **van Olmen, T.**, **Ouvrard, H.**, **Cocquebert, C.**, **Frogé, B.**, **Haritatos, A.**, **Nikas, K.**, **Gkountouvas, T.**, **Papadopoulou, N. and Athanasaki, E.** (2013). *Best Practice Guide – Curie*, `https://doi.org/10.5281/zenodo.6534658`.

[49] *TOP500 Supercomputer sites*, `http://top500.org/`, accessed Jan. 2016.

[50] *The Green500 List*, `https://www.top500.org/lists/green500/`, accessed Jan. 2016.

[51] *OpenFOAM main website*, `http://www.openfoam.com`, accessed Jan. 2016.

[52] **Sao, P.**, **Vuduc, R. and Li, X.S.** (2014). A distributed CPU-GPU sparse direct solver, *Euro-Par 2014 Parallel Processing: 20th International Conference, Porto, Portugal, August 25-29, 2014. Proceedings 20*, Springer, pp.487–498.

[53] **Celebi, M.S.**, **Duran, A.**, **Tuncel, M. and Akaydin, B.** (2012). PRACE-2IP white paper: Scalable and improved SuperLU on GPU for heterogeneous systems, **Technical Report: 283493**, Department of Mathematical Engineering, Istanbul Technical University, `https://prace-ri.eu/wp-content/uploads/scalablesuperluongpu.pdf`.

[54] **Li, X.S. and Demmel, J.W.** (2003). SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems, *ACM Transactions on Mathematical Software (TOMS)*, *29*(2), 110–140.

[55] **Amestoy, P.R.**, **Duff, I.S.**, **L'Excellent, J.Y. and Koster, J.** (2001). A fully asynchronous multifrontal solver using distributed dynamic scheduling, *SIAM Journal on Matrix Analysis and Applications*, *23*(1), 15–41.

[56] **Schenk, O. and Gärtner, K.** (2002). Solving unsymmetric sparse systems of linear equations with PARDISO, *Computational Science—ICCS 2002: International Conference Amsterdam, The Netherlands, April 21–24, 2002 Proceedings, Part II*, Springer, pp.355–363.

[57] **Schenk, O. and Gärtner, K.** (2006). On fast factorization pivoting methods for sparse symmetric indefinite systems, *Electronic Transactions on Numerical Analysis*, *23*(1), 158–179.

[58] **Duran, A. and Saunders, B.** Gen_SuperLU package (version 1.0, August 2002), referenced as GSLU also, a part of LinBox package, *GSLU contains a set of subroutines to solve a sparse linear system A* X= B over any field*.

[59] **Duran, A.**, **Saunders, B.D. and Wan, Z.** (2003). Hybrid algorithms for rank of sparse matrices, *Proceedings of the SIAM International Conference on Applied Linear Algebra (SIAM-LA)*, pp.15–19.

[60] **Duran, A.**, **Celebi, M.S.**, **Tuncel, M. and Akaydin, B.** (2012). *Design and Implementation of New Hybrid Algorithm and Solver on CPU For Large Sparse Linear Systems*, `https://doi.org/10.5281/zenodo.810699`, project final report.

[61] **Marchenko, V.A. and Pastur, L.A.** (1967). Distribution of eigenvalues for some sets of random matrices, *Matematicheskii Sbornik*, *114*(4), 507–536.

[62] **Strang, G.** (2006). *Linear algebra and its applications.*, Belmont, CA: Thomson, Brooks/Cole.

[63] **Hernandez, V.**, **Roman, J.E. and Vidal, V.** (2005). SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems, *ACM Transactions on Mathematical Software (TOMS)*, *31*(3), 351–362.

[64] **Balay, S.**, **Buschelman, K.**, **Gropp, W.D.**, **Kaushik, D.**, **Knepley, M.G.**, **McInnes, L.C.**, **Smith, B.F. and Zhang, H.** (2001). PETSc, *See http://www.mcs.anl.gov/petsc*.

[65] **nPartition Administrator's Guide, H.** part number: 5991-1247B, February 2007, *Hewlett-Packard Development Company*.

[66] *National Center for High Performance Computing (UHeM), Technical Spesifications of Karadeniz Server*, `http://www.uybhm.itu.edu.tr`, accessed June 2014.

[67] **Duran, A.**, **Celebi, M.S.**, **Tuncel, M. and Oztoprak, F.** (2013). *Structural Analysis of Large Sparse Matrices for Scalable Direct Solvers*, https://doi.org/10.5281/zenodo.831525, project final report.

[68] **Duran, A.**, **Celebi, M.S.**, **Tuncel, M. and Akaydin, B.** Scalability of SuperLU solvers for large scale complex reservoir simulations, *SPE and SIAM Conference on Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs, Istanbul, Turkey, September 3-5, 2012. SPE Conference, 2012*.

[69] *(Par)METIS homesite*, http://www.lrz.de/services/software/mathematik/metis, accessed June 2014.

[70] *Intel Optimized Math Library for Numerical Computing*, http://software.intel.com/en-us/intel-mkl, accessed June 2014.

[71] **Duran, A.**, **Celebi, M.S.**, **Piskin, S. and Tuncel, M.** (2014). *Scalability of OpenFOAM for Bio-medical Flow Simulations*, https://doi.org/10.5281/zenodo.822968, project final report.

[72] *BLAS (Basic Linear Algebra Subprograms)*, http://www.netlib.org/blas, accessed June 2014.

[73] **Feigenbaum, E.** (1992). *Expert systems: principles and practice*, KSL-91-79.

APPENDICES

APPENDIX A : Simulation Results
APPENDIX B : Fundamental Concepts

APPENDIX A : Simulation Results

The following six tables show our simulation results including NLS errors in Tables A.1 - A.2, MIF values in Tables A.3 - A.4, and QN iterations in Tables A.5 - A.6, according to Algorithm 1.

Table A.1 : Converged average NLS error values for Dsc time series group in order to compare grid and random approaches via simulation results.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Dsc_01	grid	0.0209	**0.0197**	0.0227	0.0242	0.0240
	random	0.0276	0.0227	**0.0202**	0.026	0.0218
Dsc_02	grid	0.0251	**0.0238**	0.0243	0.0248	0.0238
	random	0.0305	0.0299	0.0240	0.0248	**0.0235**
Dsc_03	grid	0.0171	**0.0166**	0.0205	0.0201	0.0225
	random	0.0249	0.0198	**0.0163**	0.0245	0.0215
Dsc_04	grid	0.0195	**0.0190**	0.0254	0.0260	0.0264
	random	0.0246	**0.0192**	0.0205	0.0259	0.0223
Dsc_05	grid	0.0196	**0.0189**	0.0252	0.0260	0.0251
	random	0.0266	0.0233	0.0217	0.0244	**0.0216**
Dsc_06	grid	0.0203	**0.0196**	0.0289	0.0283	0.0289
	random	0.0272	0.0234	**0.0209**	0.0268	0.0257
Dsc_07	grid	0.0220	0.0208	**0.0189**	0.0212	0.0212
	random	0.0329	0.0277	0.0224	0.0272	**0.0221**
Dsc_08	grid	0.0268	0.0263	0.0244	0.0252	**0.0235**
	random	0.0341	0.0313	**0.0252**	0.0284	0.0258
Dsc_09	grid	0.0261	0.0252	0.0275	0.0259	**0.0237**
	random	0.0315	0.0305	0.0245	0.0282	**0.0234**
Dsc_10	grid	0.0239	0.0238	**0.0227**	0.0248	0.0234
	random	0.0305	0.0274	**0.0231**	0.0265	0.0239
Dsc_11	grid	0.0232	**0.0212**	0.0217	0.0241	0.0227
	random	0.0314	0.0252	0.0245	0.0260	**0.0218**
Dsc_12	grid	0.0289	0.0272	0.0253	0.0266	**0.0246**
	random	0.0340	0.0286	0.0259	0.0308	**0.0256**
Dsc_13	grid	0.0239	**0.0221**	0.0224	0.0245	0.0221
	random	0.0333	0.0285	0.0227	0.0254	**0.0219**
Dsc_14	grid	0.0289	0.0279	**0.0274**	0.0280	0.0275
	random	0.0427	0.0335	0.0307	0.0323	**0.0273**
Dsc_15	grid	0.0258	0.0251	0.0238	0.0251	**0.0232**
	random	0.0362	0.032	0.0263	0.0270	**0.0257**
Dsc_16	grid	0.0179	**0.0172**	0.0279	0.0254	0.0268
	random	0.0220	**0.0181**	0.0187	0.0254	0.0231
Dsc_17	grid	0.0174	**0.0169**	0.0203	0.0223	0.0223
	random	0.0246	0.0211	**0.0189**	0.0237	0.0195
Dsc_18	grid	0.0209	**0.0195**	0.0242	0.0245	0.0255
	random	0.0283	0.0242	**0.0216**	0.0251	0.0229
Dsc_19	grid	0.0214	**0.0204**	0.0222	0.0226	0.0219
	random	0.0269	0.0228	**0.0215**	0.0257	0.0220
Dsc_20	grid	0.0227	**0.0216**	0.0235	0.0221	0.0226
	random	0.0296	0.0239	0.0239	0.0246	**0.0222**

Table A.2 : Comparison of grid and random approaches via simulation results with respect to average NLS error values for Prm time series group.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Prm_01	grid	0.0208	**0.0197**	0.0239	0.0241	0.0234
	random	0.0254	0.0229	**0.0186**	0.0244	0.0226
Prm_02	grid	0.0204	**0.0194**	0.0213	0.0237	0.0240
	random	0.0283	0.0249	**0.0216**	0.0257	0.0223
Prm_03	grid	0.0191	**0.0184**	0.023	0.0242	0.0255
	random	0.0249	0.0215	**0.0185**	0.0249	0.0237
Prm_04	grid	0.0207	**0.0181**	0.0236	0.0248	0.0249
	random	0.0267	0.0249	0.0227	0.0254	**0.0214**
Prm_05	grid	0.0284	0.0272	0.0262	0.0254	**0.0244**
	random	0.0364	0.0331	**0.0278**	0.0293	0.0282
Prm_06	grid	0.0197	**0.0194**	0.0238	0.0229	0.0229
	random	0.0240	0.0220	**0.0204**	0.0240	0.0240
Prm_07	grid	0.0264	0.0265	0.0250	0.0245	**0.0239**
	random	0.0350	0.0280	**0.0243**	0.0294	0.0243
Prm_08	grid	0.0249	0.0229	**0.0222**	0.0246	0.0241
	random	0.0333	0.0327	0.0261	0.0265	**0.0244**
Prm_09	grid	0.0258	**0.0242**	0.0249	0.0251	0.0249
	random	0.0311	0.0294	0.0270	0.0285	**0.0238**
Prm_10	grid	0.0211	**0.0202**	0.0244	0.0243	0.0247
	random	0.0313	0.0231	**0.0224**	0.0242	0.0225
Prm_11	grid	0.0228	**0.0212**	0.0284	0.0276	0.0284
	random	0.0284	0.0244	**0.0221**	0.0269	0.0242
Prm_12	grid	0.0242	**0.0233**	0.0254	0.0252	0.0233
	random	0.0349	0.0307	0.0267	0.0267	**0.0257**
Prm_13	grid	0.0234	0.0233	0.0222	0.0235	**0.0219**
	random	0.0348	0.0292	0.0261	0.0291	**0.0244**
Prm_14	grid	0.0183	**0.0176**	0.0252	0.0251	0.0268
	random	0.0253	0.0245	**0.0189**	0.0241	0.0223
Prm_15	grid	0.0313	0.0313	0.0296	0.0297	**0.0279**
	random	0.0438	0.0339	0.0291	0.0311	**0.0277**
Prm_16	grid	0.0184	**0.0173**	0.0255	0.0248	0.0277
	random	0.0231	0.0216	**0.0195**	0.0258	0.0233
Prm_17	grid	0.0322	0.0309	0.0299	0.0299	**0.0281**
	random	0.0415	0.0319	0.0339	0.0318	**0.0316**
Prm_18	grid	0.022	**0.0211**	0.0211	0.0213	0.0218
	random	0.0336	**0.0225**	0.0227	0.0251	0.0226
Prm_19	grid	0.0229	**0.0220**	0.0240	0.0234	0.0225
	random	0.0309	0.0278	0.0246	0.0257	**0.0227**
Prm_20	grid	0.0192	**0.0187**	0.0215	0.0239	0.0209
	random	0.0268	0.0246	0.0220	0.0221	**0.0215**

Table A.3 : Resulting average MIF values for Dsc time series group for comparison of grid and random approaches via simulation results.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Dsc_01	grid	0.2760	0.2764	0.2458	0.2649	**0.2339**
	random	0.3544	0.3605	**0.2653**	0.3664	0.2879
Dsc_02	grid	0.2765	0.2689	0.2425	0.2546	**0.2295**
	random	0.3839	0.3695	**0.2548**	0.2854	0.2958
Dsc_03	grid	0.2840	0.2844	0.2453	0.2555	**0.2211**
	random	0.3435	0.3746	**0.2439**	0.3594	0.2825
Dsc_04	grid	0.2898	0.2941	0.2526	0.2774	**0.2467**
	random	0.3633	0.3425	**0.2576**	0.3711	0.3070
Dsc_05	grid	0.2730	0.2749	0.2422	0.2699	**0.2358**
	random	0.3606	0.3464	**0.2584**	0.3470	0.2932
Dsc_06	grid	0.2829	0.2819	0.2538	0.2676	**0.2439**
	random	0.3519	0.3442	**0.2533**	0.4058	0.3011
Dsc_07	grid	0.2773	0.2711	**0.2251**	0.2543	0.2315
	random	0.4186	0.3983	**0.2485**	0.3238	0.2798
Dsc_08	grid	0.3005	0.2918	0.2580	0.2615	**0.2445**
	random	0.4373	0.4204	**0.2502**	0.3097	0.2987
Dsc_09	grid	0.2836	0.2807	0.2541	0.2498	**0.2172**
	random	0.4313	0.4024	**0.2639**	0.3133	0.2944
Dsc_10	grid	0.3011	0.2911	0.2543	0.2868	**0.2515**
	random	0.4274	0.3964	**0.2661**	0.3441	0.2812
Dsc_11	grid	0.2847	0.2649	**0.2379**	0.2644	0.2444
	random	0.3937	0.3501	**0.2810**	0.3054	0.2823
Dsc_12	grid	0.2750	0.2741	0.2413	0.2564	**0.2322**
	random	0.3876	0.3540	**0.2412**	0.3164	0.2702
Dsc_13	grid	0.2752	0.2480	**0.2177**	0.2471	0.2261
	random	0.4215	0.3569	**0.2406**	0.3086	0.2553
Dsc_14	grid	0.2632	0.2550	0.2361	0.2522	**0.2170**
	random	0.4481	0.3803	**0.2580**	0.2883	0.2703
Dsc_15	grid	0.2856	0.2731	0.2450	0.2513	**0.2123**
	random	0.4324	0.4094	**0.254**	0.2948	0.2752
Dsc_16	grid	0.2607	0.2699	0.2483	0.2527	**0.2276**
	random	0.3465	0.3378	**0.2620**	0.4092	0.2987
Dsc_17	grid	0.2882	0.2913	0.2705	0.2785	**0.2388**
	random	0.3725	0.3664	**0.2487**	0.3648	0.2965
Dsc_18	grid	0.2726	0.2712	0.2438	0.2658	**0.2366**
	random	0.3603	0.3409	**0.2698**	0.3676	0.2918
Dsc_19	grid	0.2723	0.2757	0.2427	0.2545	**0.2384**
	random	0.3662	0.3511	**0.2364**	0.3375	0.2849
Dsc_20	grid	0.2849	0.2808	**0.2419**	0.2675	0.2439
	random	0.3757	0.3745	**0.2858**	0.3509	0.2858

Table A.4 : Resulting average MIF values for Prm time series group for comparison of grid and random approaches via simulation results.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Prm_01	grid	0.2859	0.2896	0.2629	0.2729	**0.2360**
	random	0.3462	0.3641	**0.2571**	0.3519	0.2976
Prm_02	grid	0.2944	0.2840	0.2510	0.2769	**0.2459**
	random	0.4014	0.3872	**0.2666**	0.3453	0.3130
Prm_03	grid	0.2886	0.2891	0.2564	0.2728	**0.2519**
	random	0.3500	0.3701	**0.2426**	0.3683	0.3017
Prm_04	grid	0.2829	0.2642	**0.2305**	0.2661	0.2323
	random	0.3796	0.3620	**0.2572**	0.3559	0.2880
Prm_05	grid	0.2957	0.2844	0.2586	0.2653	**0.2418**
	random	0.4520	0.4098	**0.2702**	0.3073	0.3104
Prm_06	grid	0.2904	0.2928	0.2552	0.2541	**0.2179**
	random	0.3642	0.3495	**0.2472**	0.3488	0.2918
Prm_07	grid	0.2857	0.2839	0.2614	0.2393	**0.2213**
	random	0.4267	0.3830	**0.2533**	0.3120	0.2805
Prm_08	grid	0.3006	0.2788	**0.2447**	0.2712	0.2465
	random	0.4156	0.4051	**0.263**	0.3090	0.2894
Prm_09	grid	0.2819	0.2775	0.2517	0.2580	**0.2301**
	random	0.3891	0.3795	**0.2711**	0.3087	0.2770
Prm_10	grid	0.2747	0.2739	0.2565	0.2650	**0.2363**
	random	0.3717	0.3432	**0.2525**	0.3537	0.2974
Prm_11	grid	0.3076	0.3026	0.2747	0.2741	**0.2425**
	random	0.3530	0.3445	**0.2747**	0.4123	0.3105
Prm_12	grid	0.3028	0.2944	0.2521	0.2755	**0.2517**
	random	0.4479	0.4187	**0.2839**	0.3366	0.3164
Prm_13	grid	0.2590	0.2527	0.2287	0.2373	**0.2214**
	random	0.4079	0.3802	**0.2631**	0.3193	0.2689
Prm_14	grid	0.2730	0.2683	**0.2409**	0.2672	0.2409
	random	0.3498	0.3475	**0.2493**	0.3882	0.3005
Prm_15	grid	0.2778	0.2672	0.2401	0.2471	**0.2241**
	random	0.4658	0.3739	**0.2472**	0.2782	0.2915
Prm_16	grid	0.2771	0.2759	0.2476	0.2559	**0.2324**
	random	0.3393	0.3650	**0.2588**	0.3944	0.2951
Prm_17	grid	0.2708	0.2537	0.2371	0.2559	**0.2274**
	random	0.4443	0.3731	**0.2503**	0.2866	0.2771
Prm_18	grid	0.2980	0.2889	0.2545	0.2625	**0.2364**
	random	0.4267	0.3569	**0.2653**	0.3138	0.291
Prm_19	grid	0.2957	0.2913	0.2786	0.2738	**0.2437**
	random	0.4339	0.4193	**0.2732**	0.3180	0.2823
Prm_20	grid	0.2777	0.2789	0.2529	0.2699	**0.2453**
	random	0.3735	0.3713	**0.2629**	0.3066	0.2748

Table A.5 : Average QN iteration numbers for Dsc time series group via simulation, while using grid and random approaches.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Dsc_01	grid	**160.76**	174.22	191.55	228.32	261.17
	random	**110.26**	144.85	189.43	210.17	248.76
Dsc_02	grid	**156.47**	190.43	215.96	250.63	278.19
	random	**120.29**	164.56	197.03	260.63	258.69
Dsc_03	grid	**138.65**	174.33	192.68	225.13	259.10
	random	**122.63**	155.34	198.25	218.45	244.66
Dsc_04	grid	**131.58**	151.01	174.77	211.8	239.16
	random	**116.06**	153.69	181.38	198.69	238.68
Dsc_05	grid	**149.37**	164.62	179.99	208.61	251.14
	random	**132.78**	154.96	180.07	215.44	235.97
Dsc_06	grid	**141.77**	158.81	157.67	181.85	227.76
	random	**104.41**	144.52	167.08	186.75	218.60
Dsc_07	grid	**186.29**	201.63	224.06	245.88	279.92
	random	**130.71**	151.26	183.91	230.65	254.45
Dsc_08	grid	**168.50**	177.14	205.08	233.14	279.82
	random	**133.34**	158.17	204.09	238.28	266.57
Dsc_09	grid	**156.33**	190.29	206.86	247.27	303.72
	random	**126.53**	163.96	197.68	239.13	273.95
Dsc_10	grid	**141.07**	161.97	197.20	235.39	287.87
	random	147.19	**144.39**	182.20	228.11	260.47
Dsc_11	grid	**149.48**	187.40	204.52	240.62	289.82
	random	**118.78**	183.90	173.05	237.26	257.43
Dsc_12	grid	**167.87**	185.44	211.58	239.35	280.33
	random	**137.59**	166.76	195.70	249.74	282.19
Dsc_13	grid	**168.16**	190.39	199.22	253.30	298.21
	random	**117.61**	197.18	191.91	247.85	272.91
Dsc_14	grid	**177.74**	191.29	213.69	247.1	310.91
	random	**116.57**	158.93	180.75	241.7	250.89
Dsc_15	grid	**159.45**	185.32	206.38	243.06	286.24
	random	**99.48**	143.9	187.44	241.81	272.69
Dsc_16	grid	**139.07**	160.98	162.45	205.10	219.78
	random	**115.43**	135.97	167.04	188.07	221.50
Dsc_17	grid	**145.96**	155.62	184.64	225.83	248.74
	random	**93.88**	145.69	180.41	209.03	237.80
Dsc_18	grid	**138.12**	150.62	172.60	210.64	241.11
	random	**98.37**	153.36	171.31	195.86	223.10
Dsc_19	grid	**151.77**	183.66	210.22	229.72	289.66
	random	**131.73**	155.17	197.12	213.05	265.67
Dsc_20	grid	**142.02**	162.11	188.93	236.89	270.06
	random	**129.67**	164.82	164.88	214.97	261.09

Table A.6 : Average QN iteration numbers for Prm time series group via simulation, for the pair of grid and random approaches.

Sequence name	Parameter Type	Different parameter vector lengths 256	512	1024	2048	4096
Prm_01	grid	**150.71**	159.29	178.3	223.68	250.75
	random	**105.25**	164.97	186.13	216.81	251.34
Prm_02	grid	**157.95**	169.59	197.41	218.48	265.86
	random	**107.04**	175.18	206.23	228.54	254.92
Prm_03	grid	**140.29**	167.90	184.44	214.33	236.91
	random	**116.28**	162.20	180.64	195.69	236.15
Prm_04	grid	**148.34**	180.52	184.23	214.00	255.9
	random	**125.61**	161.38	178.11	207.02	250.54
Prm_05	grid	**156.88**	184.54	208.87	240.46	292.68
	random	**139.90**	156.81	182.39	246.15	253.08
Prm_06	grid	**157.38**	166.88	173.82	219.83	273.35
	random	**123.56**	142.59	170.42	209.64	226.73
Prm_07	grid	**160.92**	185.33	207.51	246.64	293.85
	random	**134.73**	171.13	198.98	247.66	273.80
Prm_08	grid	**143.77**	183.31	211.86	221.32	282.56
	random	**138.03**	148.13	175.10	241.64	266.25
Prm_09	grid	**151.18**	157.33	185.25	228.89	273.08
	random	**138.13**	161.83	168.12	210.72	262.54
Prm_10	grid	**146.10**	170.36	183.89	220.63	251.73
	random	**115.62**	160.87	180.58	210.63	242.83
Prm_11	grid	**130.40**	149.41	152.58	198.97	229.83
	random	**100.32**	134.06	164.79	183.12	217.74
Prm_12	grid	**144.62**	164.69	191.51	218.49	268.92
	random	**136.07**	152.55	161.83	207.58	237.27
Prm_13	grid	**177.78**	192.27	218.33	242.44	325.66
	random	**131.51**	165.06	204.76	236.78	289.99
Prm_14	grid	**133.11**	162.03	160.97	214.28	239.34
	random	**116.49**	153.45	180.34	205.24	218.73
Prm_15	grid	**162.74**	199.50	224.89	257.67	295.65
	random	**100.64**	191.74	185.79	234.11	253.98
Prm_16	grid	**132.1**	154.91	165.93	203.16	229.35
	random	**97.44**	144.62	176.98	177.10	213.48
Prm_17	grid	**182.13**	206.61	220.81	251.37	304.48
	random	**144.46**	193.56	196.58	255.97	283.44
Prm_18	grid	**138.01**	174.52	206.20	235.99	279.25
	random	**110.82**	189.78	165.23	233.55	250.96
Prm_19	grid	**155.87**	174.79	186.11	222.02	282.93
	random	**104.87**	132.55	179.0	221.69	259.16
Prm_20	grid	**161.22**	195.57	208.96	240.94	281.56
	random	154.47	**153.26**	187.08	223.09	252.87

APPENDIX B : Fundamental Concepts

Here, We give brief definitions for the following fundamental concepts introduced in chapters.

CPU (Central Processing Unit) is an electronic circuitry in which arithmetic, logic, and controlling units execute given instructions. It is the main important part of computers.

GPU (Graphics Processing Unit) refers to an electronic circuitry that is able to render graphics in performance. Nowadays, GPUs are used to handle intense numerical calculations for high-performance algorithms and machine learning, as well.

CUDA (Compute Unified Device Architecture) is a computing platform for parallel algorithms that makes general-purpose computing on GPUs possible as an application programming interface.

OpenMP (Open Multi-Processing) is an application programming interface take makes multiprocessing programming possible for shared-memory systems.

MPI (Message Passing Interface) is a library that enables advanced computational devices to communicate with each other in high performance.

FLOPS (Floating Point Operations Per Second) is a metric for computers that shows the performance of the scientific computations with the number of instructions per second.

CURRICULUM VITAE

Name SURNAME: Mehmet TUNÇEL

EDUCATION:

- **B.Sc.:** 2008, Mimar Sinan Fine Arts University, Faculty of Science and Letters, Mathematics.
- **M.Sc.:** 2013, Istanbul Technical University, Informatics Institute, Computational Science and Engineering.

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2012−2019, Research Assistant, Istanbul Technical University, Informatics Institute.
- 2019−..., Lecturer, Istanbul Technical University, Artificial Intelligence and Data Science Application and Research Center (ITU AI).

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Tunçel, M.** and Duran, A. (2023). Effectiveness of grid and random approaches for a model parameter vector optimization. *Journal of Computational Science - Elsevier*, *67*, 101960. https://doi.org/10.1016/j.jocs.2023.101960. (Article Instance)
- Duran, A. and **Tunçel, M.** (2016). Evaluation of a new parallel numerical parameter optimization algorithm for a dynamical system, *Proceedings of the 2nd International Conference Numerical Computations: Theory and Algorithms (NUMTA2016), Italy, 19-25 June 2016, AIP Conference Proceedings, 1776*, 090052. https://doi.org/10.1063/1.4965416. (Presentation Instance) https://web.itu.edu.tr/aduran/B1_Duran_Tuncel_2016_NUMTA_187.pdf
- Duran, A., Piskin, S. and **Tunçel, M.** (2016). Evaluating the maturity of OpenFOAM simulations on GPGPU for bio-fluid applications, *Proceedings of the Emerging Technology (EMiT) Conference*, pp. 11-14, Barcelona Supercomputing Center, Spain, 2-3 June 2016, editors: B.D.Rogers, D.Topping, F.Mantovani, M.K.Bane. ISBN 978-0-9933426-3-9. (Presentation Instance) https://web.itu.edu.tr/aduran/B2_Duran_Piskin_Tuncel_2016_EMiT.pdf

- Duran, A. and **Tunçel, M.** (2014). Spectral effects of large matrices from oil reservoir simulators on performance of scalable direct solvers. *In SPE Large Scale Computing and Big Data Challenges in Reservoir Simulation Conference and Exhibition*. OnePetro. https://doi.org/10.2118/172984-MS. (Presentation Instance)

- Duran, A. and **Tunçel M.** (2014). Scalable parallel nonlinear parameter optimization algorithm with parameter pools, *PRACE PN: 283493, PRACE-2IP Extension, Scalable Algorithms, WP 185*, August 11, 2014. https://doi.org/10.5281/zenodo.825430. (White Paper - Project Closure Report) https://prace-ri.eu/wp-content/uploads/WP185.pdf

- Duran A. and **Tunçel M.** (2014). A report on summary of novel programming techniques results, D12.5, Section 3.3, Page 23, *PRACE (Partnership for Advanced Computing in Europe), PRACE PN:RI-283493, PRACE-2IP Extension*, August 31, 2014. https://doi.org/10.5281/zenodo.6572440. (Project Closure Report Summary) https://prace-ri.eu/wp-content/uploads/2IP-D12.5.pdf

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- Duran, A., **Tunçel, M.** and Özer, H. Ü. (2020). GPU programlama ile yüksek boyutlu yoğun matrislerin kronecker çarpımlarının hesaplanması, 2019, *Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, *36*(1), pp. 120-128, Nisan 2020.

- Duran, A., Celebi, M.S. and **Tunçel, M.** (2012-2017). Parallel algorithm (kernel) development for large scale sparse linear systems in oil reservoir simulation (PASSOR), Researcher, *Computational Linear Algebra Project for Saudi ARAMCO*, July 1, 2012-2017.

- Duran, A., Celebi, M.S., Piskin, S. and **Tunçel, M.** (2015). Scalability of OpenFOAM for bio-medical flow simulations, *Journal of Supercomputing - Springer*, *71*(3), 2015, pp. 938-951. https://doi.org/10.1007/s11227-014-1344-1.

- Celebi, M.S., Duran, A., Oztoprak, F., **Tunçel, M.** and Akaydin, B. (2016). On theimprovement of a scalable sparse direct solver for unsymmetrical linear equations, *Journal of Supercomputing - Springer*, *73*(5), 2016, pp. 1852-1904. https://doi.org/10.1007/s11227-016-1892-7.

- Duran, A., Celebi, M.S. and **Tunçel, M.** (2012). Scalability of SuperLU solvers for large scale complex reservoir simulations, *Abstract Book, Int. Conference for Mathematical Methods in Fluid Dynamics and Simulation of Giant Oil and Gas Reservoirs, SPE and SIAM Conf., Istanbul, Turkey*, Sept. 3-5, 2012.

- Duran, A., Celebi, M.S., **Tunçel, M.** and Oztoprak, F. (2014) Spectral analysis of large sparse matrices for scalable direct solvers, *Advances in Applied Mathematics, Springer Proceedings in Mathematics & Statistics*, *87*, pp. 153-160, 2014. https://doi.org/10.1007/978-3-319-06923-4_14.

- **Tunçel, M.**, Duran, A., Celebi, M.S., Akaydin, B., and Topkaya, F.O. (2016). A Comparison of SuperLU solvers on the Intel Mic architecture, *Proceedings of the 2nd International Conference Numerical Computations: Theory and Algorithms (NUMTA2016), AIP Conference Proceedings*, *1776*, 090030. https://doi.org/10.1063/1.4965394.

- Mazza, I., Duran, A., Hundur, Y., Persi, C., Santoro, A. and **Tunçel, M.** (2016). Scalability of OpenFOAM for simulations of a novel electromagnetic stirrer for steel casting, *Proceedings of the 2016 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'16), World Congress in Computer Science, Computer Engineering & Applied Computing*,pp. 111-116, ISBN: 1-60132-444-8, CSREA Press, July 25-28, 2016, Las Vegas, USA.

- Mazza, I., Duran, A., Hundur, Y., Persi, C., Santoro, A. and **Tunçel, M.** (2016). HPC-based design of a novel electromagnetic stirrer for steel segment casting, *in EU Horizon 2020 - PRACE 4IP, WP214*, April 2016. https://doi.org/10.5281/zenodo.825968. (White Paper - Project Closure Report) https://prace-ri.eu/wp-content/uploads/WP214.pdf

- Duran, A., Celebi, M.S., Piskin, S. and **Tunçel, M.** (2014). Scalability of OpenFOAM for bio-medical flow simulations, *PRACE PN: RI-312763, PRACE-3IP, Application scalability: Computational Fluid Dynamics (CFD) applications, WP 162*, June 9, 2014. https://doi.org/10.5281/zenodo.822968. (White Paper - Project Closure Report)
  https://prace-ri.eu/wp-content/uploads/WP162.pdf

- Duran, A. and **Tunçel, M.** (2014). Exploitation of HPC tools and techniques, D7.2.2, Section 3.5, *PRACE (Partnership for Advanced Computing in Europe), PRACE PN: RI-312763, PRACE-3IP*, May 24, 2014. https://doi.org/10.5281/zenodo.6575526. (Project Closure Report Summary)
  https://prace-ri.eu/wp-content/uploads/3IP-D7.2.2.pdf

- Duran, A., Celebi, M.S., Akaydin, B., **Tunçel, M.** and Oztoprak, F. (2013). Analysis of SuperLU solvers on the Intel MIC architecture, *PRACE PN: 261557, PRACE-1IP Extension, Evaluations on Intel MIC, WP 135*, December 25, 2013. https://doi.org/10.5281/zenodo.822644. (White Paper - Project Closure Report)
  https://prace-ri.eu/wp-content/uploads/wp135.pdf

- Duran, A. and **Tunçel, M.** (2013). A report on application enabiling for capability science in the MIC architecture, D7.1.3, Section 3.11, *PRACE (Partnership for Advanced Computing in Europe), PRACE PN: RI-261557, PRACE-1IP*, December 13, 2013. https://doi.org/10.5281/zenodo.6553059. (Project Closure Report Summary)
  https://prace-ri.eu/wp-content/uploads/1IP-D7.1.3.pdf

- Duran, A., Celebi, M.S., **Tunçel, M.** and Oztoprak F. (2013). Structural analysis of large sparse matrices for scalable direct solvers, *PRACE PN: 283493, PRACE-2IP, Scalable Algorithms, WP 82*, August 20, 2013. https://doi.org/10.5281/zenodo.831525. (White Paper - Project Closure Report)
  https://prace-ri.eu/wp-content/uploads/wp82.pdf

- Celebi, M.S., Duran, A., **Tunçel, M.**, Akaydin B. and Oztoprak F. (2013). Performance analysis of BLAS libraries in SuperLU_DIST for SuperLU_MCDT (Multi Core Distributed) development, *PRACE PN: 283493, PRACE-2IP, Libraries, WP 83*, July 11, 2013. https://doi.org/10.5281/zenodo.831527. (White Paper - Project Closure Report)
  https://prace-ri.eu/wp-content/uploads/wp83.pdf

- Duran, A. and **Tunçel, M.** (2013). A report on the survey of HPC tools and techniques, D7.2.1, Section 4.1, *PRACE (Partnership for Advanced Computing in Europe), PRACE PN: RI-312763, PRACE-3IP*, April 29, 2013. https://doi.org/10.5281/zenodo.6575492. (Project Closure Report Summary)
  https://prace-ri.eu/wp-content/uploads/3IP-D7.2.1.pdf

- Duran, A., Celebi, M.S., **Tunçel, M.** and Akaydın B. (2012). Design and implementation of new hybrid algorithm and solver on CPU for large sparse linear systems, PRACE (Partnership for Advanced Computing in Europe), *PRACE PN: 283493, PRACE-2IP, Libraries, WP 43*, July 13, 2012.

https://doi.org/10.5281/zenodo.810699. (White Paper - Project Closure Report)
https://prace-ri.eu/wp-content/uploads/wp43-newhybridalgorithmfo_lsls.pdf

- Celebi, M.S., Duran, A., **Tunçel, M.** and Akaydın, B. (2012). Scalable and improved SuperLU on GPU for heterogeneous systems, *PRACE (Partnership for Advanced Computing in Europe), PRACE PN: 283493, PRACE-2IP, Libraries, WP 44*, July 13, 2012. https://doi.org/10.5281/zenodo.815126. (White Paper - Project Closure Report)
https://prace-ri.eu/wp-content/uploads/scalablesuperluongpu.pdf