

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
DEPARTAMENTO DE INFORMÁTICA
VALPARAÍSO - CHILE



“COMPLEX STEP: A DIFFERENT PERSPECTIVE ON
SOLVING OVER-DETERMINED NON-LINEAR SYSTEMS
OF EQUATIONS”

RUSSEL ANDRÉ GUEVARA HUALLPA

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERO CIVIL EN INFORMÁTICA

Profesor Guía: Claudio E. Torres, Ph.D.
Profesor Correferente: Álvaro Salinas R., M.Sc.

Agosto - 2025



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título; ☐ Tesis de Postgrado;

Título del trabajo: Complex Step: A Different Perspective on Solving Over-Determined Non-Linear Systems of Equations

Nombre del candidato(a): Russel André Guevara Huallpa

Carrera / Grado: Ingeniero Civil en Informática

Campus: Casa Central Valparaíso; **Departamento:** Departamento de Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Claudio E. Torres, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL

☒ El trabajo **NO contiene información que amerite confidencialidad** y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (embargo) por:

☐ 6 meses; ☐ 12 meses; ☐ 2 años; ☐ 3 años; ☐ 5 años; ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 04-09-2025

Firma:

Claudio E. Torres

Estudiante o Candidato(a):

Fecha: 04-09-2025

Firma:

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

DEDICATORIA

Dedicado a mi familia, a quienes tal vez nunca pueda devolver todo el amor que me han
dado.

AGRADECIMIENTOS

Agradezco de corazón a todas las personas que me han ayudado a ver mejor el horizonte, eliminando la bruma.

Agradezco enormemente al profesor Claudio Torres, por su paciencia, por su simpatía, y sobretodo por despertar en mí el deseo de aprender con el único fin de satisfacer mi curiosidad.

RESUMEN

Resumen— Uno de los principales retos en varias áreas de la ingeniería y ciencias es resolver sistemas de ecuaciones no lineales sobredeterminados, en gran parte debido a las altas demandas de cómputo y memoria necesarias para calcular la matriz Jacobiana. Este trabajo se centra en un marco de trabajo que utiliza el algoritmo no-simétrico LSQR (ns-LSQR). El núcleo de nuestra propuesta aborda la aproximación de los productos Jacobiano-vector (JVP) y Jacobiano-traspuesto-vector (JTVP). Para el JVP, se emplea el método de Complex Step, demostrando su superioridad en precisión y estabilidad numérica frente a las diferencias finitas. Para el JTVP, se evidencia que las estrategias de aproximación triviales pueden llevar al colapso del subespacio de Krylov, por lo que se propone y valida la Heurística Aleatoria, comparándola con las Heurísticas Siempre Mejor y Vanilla. Este método utiliza un subconjunto aleatorio de vectores base canónicos o de bases wavelet para construir una aproximación robusta del JTVP. Los experimentos numéricos validan que la Heurística Aleatoria previene el colapso del subespacio de Krylov, permitiendo la convergencia del método ns-LSQR incluso con un presupuesto computacional limitado. Además, se demuestra que el uso de una base wavelet puede acelerar la convergencia en problemas específicos al ofrecer una representación más dispersa del JTVP.

Palabras Clave— Sistema de ecuaciones no lineales, Matriz Jacobiana, Complex Step, ns-LSQR, Heurística Aleatoria, Heurística Siempre Mejor, Heurística Vanilla.

ABSTRACT

Abstract— One of the main challenges in various areas of engineering and science is solving over-determined non-linear systems of equations, largely due to the high computational and memory demands required to compute the Jacobian matrix. This work focuses on a framework that employs the non-symmetric LSQR (ns-LSQR) algorithm. The core of our proposal addresses the approximation of the Jacobian-vector product (JVP) and the Jacobian-transposed-vector product (JTVP). For the JVP, the Complex Step method is used, demonstrating its superior accuracy and numerical stability compared to finite differences. For the JTVP, it is shown that naive approximation strategies can lead to the collapse of the Krylov subspace. To overcome this, the Random Heuristic is proposed and validated, comparing it against the Always Best Heuristic and Vanilla Heuristic. This method uses a random subset of canonical or wavelet basis vectors to construct a robust JTVP

approximation. Numerical experiments validate that the Random Heuristic prevents the collapse of the Krylov subspace, enabling the ns-LSQR method to converge even with a limited computational budget. Furthermore, it is demonstrated that using a wavelet basis can accelerate convergence in specific problems by providing a sparser representation of the JTV.

Keywords— Non-linear system of equations, Jacobian Matrix, Complex Step, ns-LSQR, Random Heuristic, Always Best Heuristic, Vanilla Heuristic.

Glossary

CWT: Continuous Wavelet Transform

DWT: Discrete Wavelet Transform

IDWT: Inverse Discrete Wavelet Transform

JTVP: Jacobian-Transposed-Vector-Product

JVP: Jacobian-Vector-Product

LM: Levenberg-Marquardt

LSQR: An iterative algorithm for solving sparse linear equations and sparse least-squares problems

ns-LSQR: Non-symmetric LSQR

Table of contents

RESUMEN	IV
ABSTRACT	IV
Glossary	VI
List of figures	IX
Introduction	1
Chapter 1: Problem definition	3
1.1 Objectives	4
1.1.1 General objective	4
1.1.2 Specific objectives	4
Chapter 2: Literature review	5
2.1 Over-determined non-linear systems	5
2.2 Iterative methods	6
2.3 Newton's method	6
2.4 Quasi-Newton method	7
2.5 Gauss-Newton method	7
2.6 Levenberg-Marquardt method	8
2.7 The challenge of the Jacobian in large-scale problems	8
2.8 Krylov subspace methods	9
2.9 LSQR	9
2.10 Non-symmetric LSQR: ns-LSQR	10
Chapter 3: Proposed solution	12
3.1 The subspace problem	15
3.2 Main Proposal: Approximation with random canonical vectors	17
3.3 Using wavelet bases for compact representation	18
3.3.1 Wavelet transforms	19
3.3.2 Wavelet basis for the vector space	20
3.4 Jacobian transposed vector product approximation using a wavelet basis	21
3.5 Heuristics for comparison	22
Chapter 4: Numerical experiments	23
4.1 Selected functions and their jacobians	23
4.2 Basis Comparison for the JTVP	25
4.3 Krylov subspace	27
4.4 ns-LSQR heuristic comparison	28
4.4.1 The fixed-heuristic problem	30
4.4.2 Ill-conditioned jacobian	32

4.5	Finite differences vs complex-step in nsLSQR	34
4.6	Newton-nsLSQR comparison	36
4.6.1	Initial guess and distance comparison	38
Chapter 5: Conclusions		40
Chapter 6: Declaración de uso de herramientas y tecnologías de Inteligencia Artificial		42
REFERENCIAS BIBLIOGRÁFICAS		43

List of figures

1	Relative error of the JTVP for the Penalty Function II using both basis and the three heuristics	26
2	Relative error of the JTVP for the Extended Powell Singular Function using both basis and the three heuristics	26
3	Relative error of the JTVP for the Trigonometric Function using both basis and the three heuristics	27
4	Max column rank of V_k achieved by the ns-LSQR algorithm	28
5	Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm.	29
6	Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm.	29
7	Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm.	29
8	Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.	31
9	Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.	31
10	Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.	32
11	Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm (ill-conditioned case).	33
12	Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm (ill-conditioned case).	33
13	Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm (ill-conditioned case).	33
14	Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm using different finite differences methods.	34
15	Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm using different finite differences methods.	35
16	Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm using different finite differences methods.	35
17	Relative error of the Penalty Function II at each iteration of the Newton-nsLSQR algorithm.	36
18	Relative error of the Extended Powell Singular Function at each iteration of the Newton-nsLSQR algorithm.	37
19	Relative error of the Trigonometric Function at each iteration of the Newton-nsLSQR algorithm.	37
20	Residual error of the Penalty Function II at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 250$ and a coefficient percentage of 50 % for the JTVP.	38

21	Residual error of the Extended Powell Singular Function at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 470$ and a coefficient percentage of 70 % for the JTVP.	39
22	Residual error of the Trigonometric Function at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 470$ and a coefficient percentage of 70 % for the JTVP.	39

Introduction

The solution of non-linear systems of equations is a fundamental task that gives support to a vast array of applications across science and engineering. From modeling orbital mechanics [Curtis, 2019] and population dynamics [Murray, 2002] to designing complex engineering systems [Kundu y Cohen, 2010] and analyzing financial markets [Mas-Colell *et al.*, 1995], the ability to solve these systems is crucial for scientific discovery and technological advancement. A particularly challenging class of these problems involves over-determined non-linear systems, where the number of equations exceeds the number of variables. For such systems, an exact solution that satisfies all equations simultaneously often does not exist. Instead, the objective shifts to finding a best-fit solution that minimizes the discrepancy, typically formulated as a non-linear least-squares problem.

While numerous iterative methods have been developed to tackle these problems, many classical approaches, such as the Gauss-Newton [Dennis y Schnabel, 1996] and Levenberg-Marquardt [Marquardt, 1963] methods, rely on the computation of the Jacobian matrix. For small to medium-sized problems, forming this matrix is feasible. However, in the context of large-scale systems, where the number of variables and equations can range from thousands to millions, the explicit computation, storage, and manipulation of the Jacobian become prohibitively expensive, creating a significant computational bottleneck that make these methods impractical.

This limitation has motivated the development of matrix-free methods that avoid forming the Jacobian altogether. These techniques typically rely on iterative solvers that only require the ability to compute the product of the Jacobian (and its transpose) with a vector. This work is centered on this paradigm, proposing a novel and efficient matrix-free framework for solving large-scale, over-determined non-linear systems. The core of our approach is the non-symmetric LSQR (ns-LSQR) algorithm [Sanhueza, 2021], a Krylov subspace method that iteratively builds a solution without ever needing the explicit Jacobian.

The primary contributions of this work are two¹. First, we leverage the Complex Step method for computing the Jacobian vector product (JVP). We demonstrate that this technique offers superior accuracy and numerical stability compared to traditional finite difference approximations, effectively eliminating the problematic step size dependency. Second, and most critically, we address the challenge of computing the Jacobian-transposed vector product (JTVP). We first illustrate how naive approximation strategies can lead to a collapse of the Krylov subspace, causing the solver to fail. To overcome this, we propose and validate the Random Heuristic, a simple yet robust method that uses a randomized subset of basis vectors to construct a reliable approximation. We further refine this proposal by investigating the use of different orthonormal bases, namely the canonical and wavelet bases, to explore how sparse representation can enhance computational efficiency.

¹All numerical experiments and code generated from this work can be found at <https://github.com/russel-guevara/russel-memoria>

This work is structured as follows. Chapter 1 formally defines the problem of over-determined non-linear systems of equations and discusses the inherent challenges associated with solving them at a large scale. Chapter 2 provides a comprehensive literature review of existing solution methods, from classical Newton-type algorithms to modern Jacobian-free and Krylov subspace techniques, culminating in the ns-LSQR method that forms the basis of our work. Chapter 3 details our proposed solution. It introduces the Complex Step method for the JVP and presents our core contribution: a heuristic-based approach for approximating the JTVP using randomized canonical and wavelet bases. Chapter 4 presents the results of our numerical experiments, systematically comparing the performance of our proposed heuristics against baseline methods and an ideal-case benchmark, thereby validating their effectiveness and exploring their trade-offs. Chapter 5 concludes the work by summarizing our key findings, highlighting the significance of our contributions, and suggesting directions for future research.

Chapter 1

Problem definition

A non-linear system of equations consists of two or more equations in which at least one equation is non-linear. Non-linearity arises from the way variables interact; unlike linear equations, the relationship between the variables is not simply additive and proportional. For example, it can be an interaction of a variable with itself, such as being raised to a power (e.g., x^2 or x^3), or an interaction between different variables, as seen in product terms (e.g., xy). Additionally, applying inherently non-linear functions to a variable like trigonometric functions (e.g., $\cos(x)$ or $\sin(x)$), exponential functions (e.g., e^x), or logarithmic functions ($\ln(x)$) also creates a non-linear interaction.

Non-linear systems appear extensively across various scientific and engineering disciplines. For example, in physics, they are fundamental to modeling phenomena like orbital mechanics, where gravitational forces produce trajectories governed by non-linear differential equations [Curtis, 2019]. In biology, non-linear systems describe population dynamics through models such as the Lotka-Volterra predator-prey equations or logistic growth models, which capture interactions and feedback mechanisms [Murray, 2002]. In economics, non-linear systems underpin optimization problems involving utility maximization and equilibrium analysis, where relationships between variables are rarely linear [Mas-Colell et al., 1995]. In engineering, particularly in fluid dynamics, non-linear partial differential equations such as the Navier-Stokes equations describe the complex behavior of fluid flow, turbulence, and heat transfer [Kundu y Cohen, 2010].

Given the extensive applications of non-linear systems, particularly over-determined non-linear systems, numerous solution methods have been developed. Typically, these methods require computing the Jacobian matrix and its transpose. However, as system dimensionality increases, the computational demand associated with calculating and storing the Jacobian matrix escalates significantly, becoming impractical or even prohibitive for large-scale systems. In practical scenarios, computational and storage capacities are inherently limited, requiring the use of approximations for the Jacobian matrix. Nonetheless, the accuracy and effectiveness of solutions depend critically on how these approximations are constructed. Consequently, there is a pressing need to develop efficient methods that solve large-scale over-determined non-linear systems using limited computational resources while maintaining high accuracy in the solutions.

1.1. Objectives

1.1.1. General objective

The main objective of this work is to address the solution of over-determined systems of equations through the ns-LSQR method, providing new alternatives to approximate the Jacobian-vector product (JVP) and the Jacobian-transposed-vector product (JTVP).

1.1.2. Specific objectives

- To review and analyze existing methods for solving over-determined non-linear systems of equations.
- To propose an innovative way to approximate the JVP using the Complex Step method.
- To propose an innovative way to approximate the JTVP using the Complex Step method.
- To evaluate the performance of the proposed methods through numerical experiments.

Chapter 2

Literature review

In this chapter, we provide a comprehensive overview of methods for solving over-determined non-linear systems. We begin by defining the problem class and illustrating its practical significance. We then categorize existing solution approaches into classical Newton-type methods, Jacobian-free Newton–Krylov techniques, and matrix-free Krylov least-squares (LS) solvers, culminating in the non-symmetric LSQR (ns-LSQR) algorithm. Throughout, we highlight key challenges—such as the cost of Jacobian assembly, robustness in the presence of noise, and scalability—and position ns-LSQR as a promising candidate for large-scale applications.

2.1. Over-determined non-linear systems

Formally, we define a non-linear system as

$$\mathbf{F}(\mathbf{x}) = \mathbf{0}_m,$$

where $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is a non-linear multi-variable function

$$\mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_m(\mathbf{x}) \end{bmatrix},$$

$\mathbf{0}_m$ is the zero vector in $\mathbb{R}^m$, and $\mathbf{x}$ is the vector in $\mathbb{R}^n$ at which $\mathbf{F}$ is evaluated.

We say that the system is over-determined if $n < m$, square if $n = m$, and under-determined if $n > m$.

This work focuses on over-determined non-linear systems. Due to the excess of equations, an exact solution $\mathbf{x}$ that simultaneously satisfies all equations often does not exist. In practice, the problem is reformulated from finding an exact root to finding a best-fit solution. This is typically achieved by minimizing the discrepancy between the function's output and the zero vector. The standard approach is the non-linear least-squares problem, which seeks to minimize the Euclidean norm of the residual vector $\mathbf{F}(\mathbf{x})$

$$\min_{\mathbf{x} \in \mathbb{R}^n} \frac{1}{2} \|\mathbf{F}(\mathbf{x})\|^2 = \frac{1}{2} \sum_{i=1}^m f_i(\mathbf{x})^2.$$

2.2. Iterative methods

Solving over-determined non-linear systems, particularly in their least-squares formulation, almost always requires an iterative approach. Iterative methods aim to generate a sequence of approximate solutions that converges to some minimizer (or zero) of the nonlinear system $F(x) = 0$. Starting from an initial guess x_0 , each iteration computes an update Δx_i and sets

$$x_{i+1} = x_i + \Delta x_i$$

so that Δx_i progressively reduces the residual $F(x_i)$.

2.3. Newton's method

The Newton's method is a classical algorithm for solving non-linear square systems $F(x) = 0$, where $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$ is continuously differentiable [Sauer, 1982]. Starting from an initial guess x_0 , each iteration i updates the current solution by solving the linear system

$$J(x_i)\Delta x_i = -F(x_i),$$

and updating

$$x_{i+1} = x_i + \Delta x_i.$$

However, the method presents two significant challenges. First, for the method to work effectively, the Jacobian matrix must be non-singular (i.e., has inverse) near the solution. Additionally, for large-scale problems, the calculation and storage of the Jacobian matrix can be prohibitively memory-intensive.

Second, its convergence is not guaranteed with a poor initial guess, and it may diverge. To enhance its robustness and ensure convergence from starting points far from the solution, globalization strategies can be used [Dennis Jr. y Schnabel, 1996]. For instance, in a backtracking line search [Nocedal y Wright, 2006], the update is modified to ensure a sufficient decrease in a merit function, the update becomes

$$x_{i+1} = x_i + \alpha_i \Delta x,$$

where the step length α_i is chosen to ensure sufficient decrease in a merit function.

The goal is to find a step length α_i that makes meaningful progress in minimizing the merit function. It first tries the most optimistic step, setting $\alpha_i = 1$, taking the full step. Then it checks if this full step leads to a sufficiently lower value in the merit function and satisfies a specific condition like the Armijo condition [Armijo, 1966]:

$$f(x_i + \alpha_i \Delta x) \leq f(x_i) + c\alpha_i \nabla f(x_i)^T \Delta x,$$

where $c \in (0, 1)$.

If the condition is not met, the algorithm backtracks and reduces the step length by a factor (e.g., it sets $\alpha_i = \frac{\alpha_i}{2}$) and looks again to the condition. This process continues until a value of α_i is found that satisfies the sufficient decrease condition.

To apply the principles of Newton's method to the over-determined least-squares problem, the formulation must be adapted.

2.4. Quasi-Newton method

It is a variation of Newton's Method used for solving non-linear systems of equations [Birgin *et al.*, 2003]. The main advantage it provides is that it allows the use of an approximation of the Jacobian Matrix. At each iteration, the Quasi-Newton method solves the linear system

$$B(x_i)\Delta x_i = -F(x_i),$$

where $B(x_i)$ is an approximation of the Jacobian Matrix evaluated at x_i , Δx_i is the update to the current approximation, and $F(x_i)$ is the output vector of the function evaluated at x_i .

Since this method updates and refines the approximation $B(x_i)$ iteratively as the method progresses, the method converges to the solution while using a more computationally efficient approach compared to standard Newton's Method.

A popular and widely used variant of the Quasi-Newton Method is the Broyden-Fletcher-Goldfarb-Shanno (BFGS) [Dong y Jorge, 1989]. The genius of the BFGS algorithm is that it approximates the inverse Hessian matrix instead of calculating it directly. It starts with an initial guess (often the identity matrix) and iteratively improves this approximation at each step using gradient information, allowing it to achieve convergence speeds close to Newton's Method without the high computational cost, making it one of the most popular and effective optimization algorithms in practice.

2.5. Gauss-Newton method

The Gauss-Newton method is a modification of Newton's method tailored for least-square problems [Dennis y Schnabel, 1996]. It tries to solve the least-squares problem:

$$\min_{\Delta x_i} \| -F(x_i) - J(x_i)\Delta x_i \|.$$

Whose solution is given by solving the normal equations:

$$J^T(x_i)J(x_i)\Delta x_i = -J^T(x_i)F(x_i).$$

While effective, the Gauss-Newton method can struggle if the Jacobian is rank-deficient or ill-conditioned, as the matrix becomes singular or nearly singular. Furthermore, it still involves the computation of J and J^T .

2.6. Levenberg-Marquardt method

The Levenberg-Marquardt algorithm addresses the potential instability of the Gauss-Newton method by introducing a damping parameter $\lambda \geq 0$ [Marquardt, 1963]. The solution is given by solving the equation

$$(J^T(x_i)J(x_i) + \lambda_i I_n) \Delta x_i = -J^T(x_i)F(x_i).$$

The damping parameter λ_k is adjusted at each iteration. When λ_k is small, the method behaves like the Gauss-Newton method, benefiting from its fast convergence. When λ_k is large, the method behaves like a gradient descent, ensuring convergence even when far from the solution. This adaptability makes the LM method robust and one of the most popular solvers for non-linear least-squares problems.

2.7. The challenge of the Jacobian in large-scale problems

Both the Gauss-Newton and Levenberg-Marquardt methods depend of the Jacobian Matrix J . For small to medium-sized problems, forming this matrix explicitly is feasible. However, for large-scale problems, when the number of variables n and equations m can be thousands of millions, the explicit computation and storage of the $m \times n$ Jacobian matrix become prohibitively expensive in terms of both memory and computation time.

This limitation has motivated the development of Jacobian-free methods, which avoid forming the full matrix. These methods rely on iterative solvers for the linear subproblem that only require the ability to compute the product of the Jacobian and its transpose with a vector (i.e., Jv and $J^T w$).

The core of Jacobian-Free methods is the efficient and accurate computation of Jacobian-vector products. Several techniques exist, each with its own trade-offs.

The most straightforward approach to approximate a Jacobian-vector product is using finite differences. Based on the Taylor expansion, the product Jv can be approximated with the forward difference formula

$$Jv \approx \frac{F(x + hv) - F(x)}{h},$$

where $h \in \mathbb{R}$.

A more accurate approximation is the central difference formula

$$Jv \approx \frac{\mathbf{F}(\mathbf{x} + hv) - \mathbf{F}(\mathbf{x} - hv)}{2h}.$$

However, the main drawback of finite differences is its susceptibility to numerical errors. The choice of the step size h is critical and problematic: if h is too large, errors from the Taylor series dominate; if h is too small, subtractive cancellation errors can destroy the accuracy of the result. This trade-off motivates the search for more accurate and stable methods for computing matrix-vector products.

2.8. Krylov subspace methods

Jacobian-free techniques are typically embedded within an iterative solver for the linear system that arises at each step. Krylov subspace methods are ideally suited for this purpose, as they construct a solution by building a sequence of subspaces that only require matrix-vector products.

2.9. LSQR

The LSQR method is an iterative method for solving sparse linear equations and sparse least-squares problems [Paigey, 1982]. The algorithm computes a solution $\mathbf{x}$ to either of the following problems:

- Unsymmetric equations: solve $A\mathbf{x} = \mathbf{b}$.
- Linear least square: minimize $\|A\mathbf{x} - \mathbf{b}\|_2$.

It is an iterative Krylov subspace projection method that builds a sequence of solution subspaces of small but increasing dimensions. For accomplishing this, it performs a bidiagonal decomposition of matrix A . This decomposition involves generating two orthogonal bases that span the Krylov subspaces $\mathcal{K}_k(A^T A, A^T \mathbf{b})$ and $\mathcal{K}_k(AA^T, \mathbf{b})$ using the Golub-Kahan bidiagonalization process [Golub y Kahan, 1965] (Since we are working with the Jacobian, we'll use J instead of A).

Starting with $\beta_1 \mathbf{u}_1 = \mathbf{b}$ and $\alpha_1 \mathbf{v}_1 = J^T \mathbf{u}_1$

$$\mathbf{u}_1 = \frac{\mathbf{b}}{\|\mathbf{b}\|}, \tag{1}$$

$$\mathbf{v}_1 = \frac{J^T \mathbf{b}}{\|J^T \mathbf{b}\|}, \tag{2}$$

for $i = 1, 2, \dots$ we obtain the following recurrences

$$\beta_{i+1} \mathbf{u}_{i+1} = J \mathbf{v}_i - \alpha_i \mathbf{u}_i, \quad (3)$$

$$\alpha_{i+1} \mathbf{v}_{i+1} = J^T \mathbf{u}_{i+1} - \beta_{i+1} \mathbf{v}_i, \quad (4)$$

which can be written in matrix form

$$JV_k = U_{k+1} B_k, \quad (5)$$

$$J^T U_k = V_k B_k^T + \alpha_{k+1} \mathbf{v}_{k+1} \mathbf{e}_{k+1}^T. \quad (6)$$

Notice that we can compute an approximate solution to $\mathbf{x}$ restricted to the columns of V_k and hence we have $\Delta \mathbf{x} = V_k \mathbf{c}_k$ for some $\mathbf{c}_k \in \mathbb{R}^k$.

$$\min_{\Delta \mathbf{x} \in \text{range}(V_k)} \|\mathbf{b} - J \Delta \mathbf{x}\|^2 = \min_{\mathbf{c}_k \in \mathbb{R}^k} \|U_{k+1} (\|\mathbf{b}\| \mathbf{e}_1^{(k+1)} - B_k \mathbf{c}_k)\|^2 = \min_{\mathbf{c}_k \in \mathbb{R}^k} \|\|\mathbf{b}\| \mathbf{e}_1^{(k+1)} - B_k \mathbf{c}_k\|^2$$

2.10. Non-symmetric LSQR: ns-LSQR

The non-symmetric LSQR method (ns-LSQR) is a modification of the LSQR algorithm. This modification involves using a routine to compute the matrix-vector product $J(\mathbf{x})\mathbf{v}$ and another routine to approximate $J^T(\mathbf{x})\mathbf{v}$ [Sanhueza, 2021]. It first computes the equivalent vectors of (1) and (2)

$$\tilde{\mathbf{u}}_1 = \frac{\mathbf{b}}{\|\mathbf{b}\|},$$

$$\tilde{\mathbf{v}}_1 = \frac{\tilde{J}^T \mathbf{b}}{\|\tilde{J}^T \mathbf{b}\|},$$

and the recurrences (3) and (5) as

$$J \tilde{\mathbf{v}}_k = \beta_{1,k} \tilde{\mathbf{u}}_1 + \beta_{2,k} \tilde{\mathbf{u}}_2 + \dots + \beta_{k+1,k} \tilde{\mathbf{u}}_{k+1},$$

$$\tilde{J}^T \tilde{\mathbf{u}}_{k+1} = \alpha_{1,k+1} \tilde{\mathbf{v}}_1 + \alpha_{2,k+1} \tilde{\mathbf{v}}_2 + \dots + \alpha_{k+1,k+1} \tilde{\mathbf{v}}_{k+1},$$

whose matrix forms are

$$J \tilde{V}_k = \tilde{U}_{k+1} \tilde{B}_k, \quad (7)$$

$$\tilde{J}^T \tilde{U}_{k+1} = \tilde{V}_{k+1} \tilde{B}_{k+1}, \quad (8)$$

where $\tilde{B}_k$ is no longer a bidiagonal matrix but a Hessenberg matrix of size $(k+1) \times k$, and $\tilde{B}_{k+1}$ an upper triangular $(k+1) \times (k+1)$ matrix.

Same as in the standard LSQR method, for each iteration k , if we restrict our solution $\Delta \mathbf{x}$ to the column space of $\tilde{V}_k$ and we have:

$$\min_{\Delta \tilde{\mathbf{x}} \in \text{range}(\tilde{V}_k)} \|\mathbf{b} - J \Delta \tilde{\mathbf{x}}\|^2 = \min_{\tilde{\mathbf{c}}_k \in \mathbb{R}^k} \|U_{k+1} (\|\mathbf{b}\| \mathbf{e}_1^{(k+1)} - \tilde{B}_k \tilde{\mathbf{c}}_k)\|^2 = \min_{\tilde{\mathbf{c}}_k \in \mathbb{R}^k} \|\|\mathbf{b}\| \mathbf{e}_1^{(k+1)} - \tilde{B}_k \tilde{\mathbf{c}}_k\|^2$$

This is represented in the following algorithm

Algorithm 1 General nsLSQR algorithm.

Require: A function F , a vector x , to evaluate the Jacobian matrix, a stepsize $\epsilon > 0$ to be used in the finite-difference approximation, an approximation matrix $\tilde{J}$, a vector b , a tolerance η , a number of iterations t_{inner} and t_{outer} to control the restarts, an initial guess x_0 which usually is set to be the zero vector

Ensure: An approximated solution Δx .

```

1: for  $i \leftarrow 1, \dots, t_{\text{outer}}$  do
2:    $r \leftarrow b - Jx_0$  using a finite-difference approximation.
3:    $\tilde{u}_1 \leftarrow \frac{r}{\|r\|}$ 
4:    $\tilde{v}_1 \leftarrow \frac{J^T r}{\|J^T r\|}$  using the Random Heuristic that will be shown later.
5:   for  $k$  from 1 to  $t_{\text{inner}}$  do
6:     Compute  $\tilde{u}_{k+1}$  from (4.9) using a modified Gram-Schmidt procedure.
7:     Compute  $\tilde{v}_{k+1}$  from (4.11) using a modified Gram-Schmidt procedure.
8:     Solve the least-square problem  $\min_{\tilde{c}_k \in \mathbb{R}^k} \left\| \|r\| e_1^{(k+1)} - \tilde{B}_k \tilde{c}_k \right\|^2$ , for  $\tilde{c}_k$ .
9:      $\Delta x_k \leftarrow \tilde{V}_k \tilde{c}_k$ .
10:    if the solution  $\Delta x_k$  fulfill a convergence criteria using the tolerance  $\eta$  then
11:      Returns  $\Delta x_k$ .
12:    end if
13:  end for
14:   $x_0 \leftarrow x_0 + \Delta x$ 
15: end for
16: Returns  $\Delta x_k$ .
```

Chapter 3

Proposed solution

As described above, we are trying to solve the equation given by the ns-LSQR method:

$$\min_{\tilde{\mathbf{c}}_k \in \mathbb{R}^k} \left\| \|\mathbf{b}\| \mathbf{e}_1^{(k+1)} - \tilde{B}_k \tilde{\mathbf{c}}_k \right\|^2.$$

Remember that the main point of this method is that we can use the $\tilde{J}$ and $\tilde{J}^T$ approximations for calculating the vectors $\tilde{\mathbf{v}}_{k+1}$, $\tilde{\mathbf{u}}_{k+1}$. This includes the possibility of using a direct matrix-free method for computing $J\tilde{\mathbf{v}}_k$ and $J\tilde{\mathbf{u}}_{k+1}$ (e.g. using complex step as is about to be shown). At this point, we can briefly review the two earlier requirements:

1. A procedure for computing $J\mathbf{v}$ where $\mathbf{v} \in \mathbb{R}^n$ and is an arbitrary vector.
2. A procedure for approximating $J^T \mathbf{w}$ where $\mathbf{w} \in \mathbb{R}^m$ and is an arbitrary vector.

First, we will examine the definition of complex step and then how can it be use for both requirements.

The complex-step method is a numerical technique for calculating the first derivative of a real-valued function of a real variable ($f : \mathbb{R} \rightarrow \mathbb{R}$) with surprisingly high accuracy [Martins et al., 2003]. The method's formula is

$$f'(x) \approx \frac{\text{Im}[f(x + ih)]}{h},$$

where i is the imaginary unit, h is a small real step size, and $\text{Im}[\dots]$ denotes taking the **imaginary part** of the complex result.

However, as discussed earlier, we are interested in approximating not the derivative of a single function but a set of functions ($\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$). This is, the Jacobian of the multi-variable map $\mathbf{F}$.

In order to achieve this, we first analyze the case for getting the derivative of $f : \mathbb{R}^n \rightarrow \mathbb{R}$, and then the case for $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$, both using complex-step.

For the first case, we notice that we actually want to find the gradient ∇f of a multi-variable function $f(\mathbf{x}) = f(x_1, \dots, x_n)$. So we do need to compute each partial derivative $\frac{\partial f}{\partial x_k}$ separately. To accomplish this, we can apply the complex-step method to each partial derivative by introducing the imaginary step only in the direction of the variable we are differentiating with respect to [Castro, 2025]

$$\frac{\partial f(\mathbf{x})}{\partial x_k} \approx \frac{\text{Im}[f(\mathbf{x} + ih\mathbf{e}_k)]}{h},$$

where $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{e}_k$ is the k -th standard basis vector. Hence we have

$$\nabla f(\mathbf{x}) \approx \left(\frac{\text{Im}[f(\mathbf{x} + ih\mathbf{e}_1)]}{h}, \dots, \frac{\text{Im}[f(\mathbf{x} + ih\mathbf{e}_n)]}{h} \right). \quad (9)$$

For the second case, remember that the Jacobian matrix for a function ($\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$) is a matrix where the entry $J_{kj} = \frac{\partial f_k}{\partial x_j}$. This means the j -th column of the Jacobian consists of the partial derivatives of all output components ($f_1, \dots, f_m$) with respect to the single input variable x_j . The core idea then is to compute each column of the Jacobian individually using the complex-step method

$$J_{:,j} = \frac{\text{Im}[\mathbf{F}(\mathbf{x} + ih\mathbf{e}_j)]}{h}.$$

Notice how this is similar to (3) but instead of calculating a single partial derivative for a unique function $\frac{\partial f(\mathbf{x})}{\partial x_k}$, we calculate all the partial derivatives with respect to some x_k for all the functions at once $\frac{\partial f_1(\mathbf{x})}{\partial x_k}, \dots, \frac{\partial f_m(\mathbf{x})}{\partial x_k}$.

The above can be represented by the following algorithm:

Algorithm 2 Algorithm for approximating the Jacobian Matrix.

Require: A function $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ which can evaluate complex vector inputs, a vector $\mathbf{v} \in \mathbb{R}^n$.

Ensure: An approximation of the Jacobian Matrix J .

- 1: Initialize an $m \times n$ matrix J .
 - 2: Choose a small step size h .
 - 3: **for** $j \leftarrow 1, \dots, n$ **do**
 - 4: $x_{Im} \leftarrow \mathbf{x} + ih\mathbf{e}_j$ Introduce a imaginary step to the j -th component of the input vector.
 - 5: $J_{:,j} \leftarrow \frac{\text{Im}[\mathbf{F}(x_{Im})]}{h}$ Calculate and save the j -th column of J
 - 6: **end for**
 - 7: **return** J
-

We now have a way for calculating $\tilde{J}$. It is important to notice that due to the extreme accuracy of the complex step method, the Jacobian returned by algorithm 2 will be, if not equal, very close to the original.

We are particularly interested not in computing an approximation of the Jacobian, but rather in computing an approximation of $J\mathbf{v}$ and $J^T\mathbf{w}$.

Regarding the first procedure, we present the following equation:

Let $\mathbf{F} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be a map, $\Delta\mathbf{x} \in \mathbb{R}^n$ be the vector of evaluation of the Jacobian J , and $\mathbf{v} \in \mathbb{R}^n$ be the vector you want to multiply the Jacobian by. The complex step approximation for the Jacobian-vector product is:

$$\frac{\text{Im}[\mathbf{F}(\Delta\mathbf{x} + ih\mathbf{v})]}{h} \approx J(\Delta\mathbf{x}) \cdot \mathbf{v}$$

The proof for this comes from the Taylor expansion of $\mathbf{F}$ around Δx , where $z \in \mathbb{R}$:

$$\mathbf{F}(\Delta x + zv) = \mathbf{F}(\Delta x) + z(J(\Delta x)v) + \mathcal{O}(z^2).$$

Now, let the perturbation be a complex step in the direction v :

$$\begin{aligned}\mathbf{F}(\Delta x + ihv) &= \mathbf{F}(\Delta x) + ih(J(\Delta x)v) + \mathcal{O}((ih)^2), \\ &= \mathbf{F}(\Delta x) + ih(J(\Delta x)v) + \mathcal{O}(h^2).\end{aligned}$$

Assuming $\mathbf{F}(\Delta x)$ and $J(\Delta x)v$ are real vectors, the imaginary part of the right side of the equation is dominated by $h(J(\Delta x)v)$ for small h . Hence we have

$$\frac{\text{Im}[\mathbf{F}(\Delta x + ihv)]}{h} \approx J(\Delta x)v \quad (10)$$

which fulfills the first requirement.

The equation (10) can be represented by the following algorithm:

Algorithm 3 Algorithm for approximating the JVP using the complex-step method

Require: A function F that accepts complex vector inputs, the point of evaluation x_{eval} , the vector v , and a small real step size $h > 0$.

Ensure: An approximation of $J(x_{eval})v$.

- 1: Let i be the imaginary unit.
 - 2: $\mathbf{x}_{complex} \leftarrow x_{eval} + ihv$ Compute the stepped input vector.
 - 3: $F_{complex} \leftarrow F(\mathbf{x}_{complex})$ Evaluate the function with the complex input.
 - 4: $F_{imaginary} \leftarrow \text{Im}(F_{complex})$ Isolate the imaginary part of the function's output.
 - 5: $\text{JVP} \leftarrow \frac{F_{imaginary}}{h}$ Calculate the JVP approximation.
 - 6: **return** JVP.
-

In regard to the second requirement, we present this second equation from [Sanhueza y Torres, 2017]:

Let $Q = \{q_1, q_2, \dots, q_n\}$, $q_i \in \mathbb{R}^n$ be a set of n orthonormal vectors. Due to $J^T \in \mathbb{R}^{n \times m}$ and $v \in \mathbb{R}^m$, it follows that $J^T v \in \mathbb{R}^n$. Thus we can express this product as

$$J^T v = \sum_{i=1}^n \alpha_i q_i.$$

Since the vectors in Q are orthonormal, for some $1 \leq k \leq n$ we have:

$$\begin{aligned}q_k^T J^T v &= q_k^T \left(\sum_{i=1}^n \alpha_i q_i \right), \\ (Jq_k)^T v &= \alpha_k q_k^T q_k = \alpha_k, \\ J^T v &= \sum_{i=1}^n \langle Jq_i, v \rangle q_i,\end{aligned} \quad (11)$$

(with $\langle \cdot \rangle$ representing the dot product) which satisfies the second requirement.

Notice that in (11) the product Jq_i can be computed with the equation given by (10). Thus we have

$$J^T v = \sum_{i=1}^n \left\langle \frac{\text{Im}[F(\Delta x + ihq_i)]}{h}, v \right\rangle q_i. \quad (12)$$

Keep in mind that the main idea of (10) and (12) is to avoid the computation of the Jacobian matrix. Indeed, both products can be computed in a matrix-free way and mostly all complexity lies in the evaluation of F .

Regarding equation (11), which is very computationally intensive, we can ask if somehow there is something more we can do to reduce this complexity. If we do $l < n$ sums, we are doing an approximation of $J^T v$

$$J^T v \approx \sum_{i=1}^l \langle Jq_i, v \rangle q_i, \quad (13)$$

and the accuracy of the approximation will be constrained by the q_i vectors and the number l of them included in the computation. Our goal then will be to get a decent approximation without necessarily computing all the α_i coefficients.

Both the (11) and (13) equations can be expressed in the following algorithm:

Algorithm 4 Algorithm for approximating the JTVP.

Require: An orthonormal basis $Q = \{q_1, q_2, \dots, q_n\}$, a function F , the point of evaluation x_{eval} , a vector w , the number of basis vectors to use $l \leq n$, and a small step size h .

Ensure: An approximation of $J^T(x_{eval})w$.

- 1: Let n be the dimension of the vectors.
 - 2: JTVP $\leftarrow 0 \in \mathbb{R}^n$ Initialize the approximation vector.
 - 3: **for** $i \leftarrow 1$ to l **do**
 - 4: Get the basis vector q_i from Q .
 - 5: $Jq_i \leftarrow \text{JVP}(q_i, x_{eval}, F, h)$ Compute the JVP with the algorithm 3.
 - 6: $\alpha_i \leftarrow \langle Jq_i, w \rangle$ Compute the coefficient.
 - 7: JTVP $\leftarrow \text{JTVP} + \alpha_i q_i$ Update the approximation.
 - 8: **end for**
 - 9: **return** JTVP.
-

3.1. The subspace problem

Before proceeding, it is crucial to understand why a naive approximation of the JTVP using a subset of the first l canonical vectors from the canonical basis is an inadequate approach.

If we use the first l canonical vectors e_i in equation (13), we have:

$$\begin{aligned} J^T \mathbf{v} &\approx \sum_{i=1}^l \langle J e_i, \mathbf{v} \rangle e_i, \\ &\approx \sum_{i=1}^l e_i (J e_i)^T \mathbf{v}, \\ &\approx \sum_{i=1}^l e_i e_i^T J^T \mathbf{v}, \\ &\approx \left(\sum_{i=1}^l e_i e_i^T \right) J^T \mathbf{v}, \end{aligned}$$

where this last expression can be represented as

$$\underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & \ddots & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}}_{\text{first-}l\text{-columns}} J^T \mathbf{v}.$$

This expression is equivalent to multiplying $J^T \mathbf{v}$ by a projection matrix $\left(\sum_{i=1}^l e_i e_i^T \right)$ that preserves only the first l components of the vector and sets the rest to zero.

The fundamental issue with this method is its arbitrariness. The choice to discard the final $n - l$ components is not based on their magnitude or importance to the solution. The most significant components of the vector $J^T \mathbf{v}$ could lie anywhere, and this method would discard them simply if their index is greater than l .

For example, imagine we use only the vectors $\{e_1, e_2\}$ in equation (13). We have

$$J^T \mathbf{v} \approx \langle J e_1, \mathbf{v} \rangle e_1 + \langle J e_2, \mathbf{v} \rangle e_2,$$

where we can see that this approximation projects the vector $J^T \mathbf{v}$ onto the subspace generated by $\{e_1, e_2\}$. The result is a vector where only the first two components can be non-zero, while all other are set to zero. This severely impacts the Krylov subspace generated by ns-LSQR algorithm at its very first iterations and all subsequent ones.

Remember that the process for building the basis V_k starts with the initial vector

$$\tilde{\mathbf{v}}_1 = \frac{\tilde{J}^T \mathbf{b}}{\|\tilde{J}^T \mathbf{b}\|}.$$

If the JTV approximation only produces vectors with non-zero values in the first two components, then $\tilde{\mathbf{v}}_1$ will also be limited to this two-dimensional subspace. Subsequent vectors,

$\tilde{\mathbf{v}}_{k+1}$, are derived from products of the form $\tilde{\mathbf{J}}^T \tilde{\mathbf{u}}_1$. Consequently, every new basis vector $\tilde{\mathbf{v}}_{k+1}$ generated during the process will also be confined to this same, severely restricted, subspace.

As a result, the matrix $V_k = [\tilde{\mathbf{v}}_1, \tilde{\mathbf{v}}_2, \dots, \tilde{\mathbf{v}}_k]$, whose columns are supposed to form a rich basis for the solution, will have at most a rank of two. The ns-LSQR algorithm will search for a solution update $\Delta \mathbf{x}$ within this limited subspace, but it will be unable to explore other crucial dimensions of the problem. This leads to a premature stagnation, preventing the algorithm from converging to the correct solution.

3.2. Main Proposal: Approximation with random canonical vectors

As we saw, a naive approximation of the JTV that consistently uses a fixed subset of the first l canonical vectors presents significant problems. This approach is arbitrary and fails to consider the importance or magnitude of the vector's components. The most significant components of the JTV could be located anywhere within the vector, and a fixed selection method risks discarding them simply based on their index.

Let's now suppose we select l random canonical vectors for each JTV approximation. The core idea is that by changing the subspace for the projection at each iteration, the algorithm is no longer stuck in a single, arbitrarily chosen subspace. This dynamic selection process encourages the algorithm to explore different dimensions of the problem over time.

While a single random selection might not be perfect, the key is that subsequent iterations have the chance to sample more relevant directions. This stochastic exploration helps to prevent the premature stagnation seen with the fixed selection method, offering a better opportunity to build a richer, more representative Krylov subspace that can lead to convergence.

Our central hypothesis is that a randomized projection for the JTV can overcome the subspace collapse observed in deterministic, partial approximations, enabling the ns-LSQR method to converge effectively for large-scale problems even with a limited computational budget per iteration. This will be our main proposal and we'll call it the Random Heuristic. This can be represented by the following algorithm.

Algorithm 5 Algorithm for approximating the JTVP using the Random Heuristic

Require: An orthonormal basis $Q = \{q_1, q_2, \dots, q_n\}$, a function F , the point of evaluation x_{eval} , a vector w , the number of basis vectors to use $l \leq n$, and a small step size h .

Ensure: An approximation of $J^T(x_{eval})w$.

- 1: Let n be the dimension of the vectors.
 - 2: $JTVP \leftarrow 0 \in \mathbb{R}^n$ Initialize the approximation vector.
 - 3: Let I be a set of l unique random indices selected from $\{1, 2, \dots, n\}$
 - 4: **for** each index i in I **do**
 - 5: Get the basis vector q_i from Q .
 - 6: $Jq_i \leftarrow JVP(q_i, x_{eval}, F, h)$ Compute the JVP with the algorithm 3.
 - 7: $\alpha_i \leftarrow \langle Jq_i, w \rangle$ Compute the coefficient.
 - 8: $JTVP \leftarrow JTVP + \alpha_i q_i$ Update the approximation.
 - 9: **end for**
 - 10: **return** JTVP.
-

Notice that this makes the equation (8) no longer true. We are no longer making a multiplication by a constant matrix $\tilde{J}^T$ but instead a dynamic process where each column on the left side is transformed by a random projection matrix. Instead of

$$\tilde{J}^T \tilde{U}_{k+1} = \tilde{V}_{k+1} \bar{B}_{k+1}$$

we have

$$(C_1 \tilde{J}^T \tilde{u}_1 \mid C_2 \tilde{J}^T \tilde{u}_2 \mid \dots \mid C_{k+1} \tilde{J}^T \tilde{u}_{k+1}) = \tilde{V}_{k+1} \bar{B}_{k+1}$$

where each C_i is a random projection matrix defined by I in the step 4 in algorithm 5.

3.3. Using wavelet bases for compact representation

While the random canonical vector approach solves the convergence problem, we can further ask: can we improve the efficiency of the approximation? This leads to a complementary refinement based on the principle of sparse representation. The hypothesis is that the vector $J^T v$ might not be sparse in the canonical basis, meaning its energy is distributed across many components. However, in a different orthonormal basis, its energy might be concentrated in just a few significant coefficients. If such a basis exists, a random sample of l vectors would have a statistically higher chance of capturing these few important coefficients, potentially leading to faster convergence or better accuracy for the same computational cost.

3.3.1. Wavelet transforms

Wavelet transforms are a powerful mathematical tool for analyzing signals and images, offering a unique ability to simultaneously examine both the frequency content of a signal and where that content occurs in time. This capability sets them apart from traditional methods like the Fourier transform and has made them indispensable in a wide array of fields, from signal and image processing to data compression and financial analysis [Burrus et al., 1998].

At its core, a wavelet transform breaks down a signal into a set of basis functions called wavelets. These are small, wave-like oscillations that are localized in time, meaning they have a distinct beginning and end. They can be scaled and combined to represent any signal or function of interest. These wavelets are generated from the mother wavelet, denoted as $\psi(t)$, through two fundamental operations:

- **Dilation (Scaling):** The mother wavelet is stretched or compressed by a scale factor, s becoming $\psi(\frac{t}{s})$. A large scale factor corresponds to a stretched wavelet, which is effective at analyzing low-frequency components of the signal. A small scale factor results in a compressed wavelet, ideal for capturing high-frequency, transient details.
- **Translation (Shifting):** The scaled wavelet is then shifted in time by a factor, u , resulting in $\psi(\frac{t-u}{s})$. This allows the analysis to be localized to specific points along the signal, revealing how its frequency content changes over time.

By systematically applying all possible dilations and translations, a complete set of wavelet basis functions is created. Each function in this set is uniquely defined by its scale and position, and together they can be used to decompose and reconstruct a signal with high fidelity. An interesting property of this wavelet basis is its orthogonality (this will depend on the chosen mother wavelet), that will be used for our desired purpose: compute the $J^T v$ product.

Wavelet transforms come in two main varieties, each suited for different analytical tasks:

- **Continuous Wavelet Transform (CWT):** In the CWT, the scaling and shifting operations occur continuously over a range of values. This produces a highly detailed and often redundant representation of the signal, making it ideal for in-depth analysis and visualization of how frequencies evolve over time. It is particularly useful for identifying transient events or subtle changes in a signal. While excellent for this type of works, its redundancy makes it computationally inefficient and unsuitable for our goal of creating a compact vector representation.
- **Discrete Wavelet Transform (DWT):** The DWT uses a more limited set of discrete scales and shifts, typically based on powers of two. This results in a more compact and efficient representation of the signal, making it the preferred choice for applications like

data compression, denoising, and numerical analysis. The DWT breaks the signal into approximation and detail coefficients, which correspond to the low-frequency and high-frequency components of the signal, respectively.

The DWT's ability to provide a compact, non-redundant, and orthonormal basis makes it the ideal choice for our application. It offers a practical method to transform the JTVP vector into a new domain where its most important components are isolated in a few coefficients, directly addressing the shortcomings of the canonical basis.

3.3.2. Wavelet basis for the vector space

The previous section introduced the DWT as the right tool for our purpose. However, a wavelet basis is conceptually a basis of functions and, for our use in our numerical algorithm, we need a concrete set of orthonormal basis vectors in $\mathbb{R}^n$. The key lies in the relationship between the DWT and its inverse: the Inverse Discrete Wavelet Transform (IDWT). The first one analyzes a vector and computes a set of coefficients that represent that vector in the wavelet basis, while the second synthesizes a vector from a given set of wavelet coefficients.

Therefore, to generate a single wavelet basis vector, we can provide the IDWT with a set of coefficients where only one coefficient is non-zero. The simplest way to represent this is with a canonical basis vector e_i . Applying the IDWT to e_i is equivalent to obtain the vector in $\mathbb{R}^n$ which corresponds to the signal composed of 100 % of the i -th wavelet basis function and 0 % of all the others. The result of this operation is the i -th wavelet basis vector itself, denoted as ψ_i .

By repeating this process for every canonical vector from e_1 to e_n , we can construct the complete orthonormal wavelet basis $W = \{\psi_1, \psi_2, \dots, \psi_n\}$ for $\mathbb{R}^n$. This can be expressed in the following algorithm.

Algorithm 6 Compute the wavelet basis for $\mathbb{R}^n$

Require: A number n that indicates the dimension of the wavelet basis,

Ensure: A wavelet basis for $\mathbb{R}^n$.

```

1:  $W \leftarrow []$  Initialize an empty list
2: for  $j \leftarrow 1, \dots, n$  do
3:    $\psi \leftarrow IDWT(e_j)$  Apply the IDWT to the canonical vector  $j$ .
4:    $W \leftarrow \psi$  Add the vector to the  $W$  list.
5: end for
6: return  $W$ 

```

We do not give a procedure or algorithm for computing the IDWT of a given vector because it goes beyond the scope of this work and, depending on the used library, it can vary. We'll

be using the PyWavelets library (an open source wavelet transform software for Python that combines a simple high-level interface with low-level C and Cython performance) for the IDWT and DWT when needed.

It's important to note that explicitly generating and storing this entire basis requires $O(n^2)$ memory, which can be prohibitive for large-scale problems. An alternative approach is to compute each basis vector ψ_i on the fly whenever it is needed, trading increased computation time for significantly reduced memory usage.

Specifically, for the numerical experiments in this work, the Haar wavelet was chosen as the mother wavelet. Its unique properties make it particularly suitable for the task of sparse representation in a computational context. The Haar transform is the simplest and fastest wavelet transform, resulting in a minimal computational overhead. In an iterative algorithm like ns-LSQR, where the JTVP is approximated repeatedly, this speed is a significant advantage that minimizes the overall execution time. While smoother wavelets like sym4 are better for representing smooth signals, the goal here is not signal fidelity but achieving the sparsest possible representation to make the random sampling heuristic more effective. The added complexity and larger support of smoother wavelets do not necessarily confer an advantage for this specific application and come at a higher computational cost. This makes even more sense if we consider computing these vectors on the fly to save storage space.

3.4. Jacobian transposed vector product approximation using a wavelet basis

By substituting the generic orthonormal basis with a wavelet basis $W = \{\psi_1, \psi_2, \dots, \psi_n\}$ for $\mathbb{R}^n$, the equation (13) becomes

$$J^T \mathbf{v} \approx \sum_{i=1}^l \langle J\psi_i, \mathbf{v} \rangle \psi_i. \quad (14)$$

Note that each coefficient is given by $\alpha_i = \langle J\psi_i, \mathbf{v} \rangle$, where the term $J\psi_i$ can be computed efficiently and in a matrix-free manner using the complex-step method.

By generating an orthonormal wavelet basis we can apply the same random sampling strategy. The potential advantage is that each random sample is more likely to yield a high-impact coefficient, thus creating a more accurate JTVP approximation with the same number of JVP computations. This approach does not change the core random sampling strategy but seeks to optimize its efficiency by choosing a more suitable underlying basis.

3.5. Heuristics for comparison

To validate our proposal, we will compare the following three heuristics for approximating the JTVP within the ns-LSQR algorithm:

- **Fixed Heuristic (Vanilla Method):** Uses the first l canonical vectors. This is expected to fail for $l < n$ and serves to demonstrate the problem described in Section 3.1.
- **Random Heuristic:** Uses l randomly selected basis vectors at each iteration. This will be tested with both the canonical basis (our main proposal) and a wavelet basis (the refinement).
- **Always Best Heuristic:** Computes all n coefficients, sorts them by magnitude, and selects the l largest. This is computationally impractical and serves as a theoretical benchmark. It is designed to simulate a hypothetical, ideal, scenario where one could know in advance which l basis vectors would yield the most significant coefficients for the JTVP approximation. By computing all coefficients and subsequently discarding the $n - l$ smallest ones, this heuristic provides the ideal benchmark for the best possible approximation accuracy achievable for a given budget of l vectors.

Chapter 4

Numerical experiments

In this section, we report and examine the findings from our numerical experiments.

The experiments are the following:

- **JTVP Basis and Heuristic Comparison:** This experiment compares the Random, Always Best, and Fixed heuristics using both canonical and wavelet bases to verify which combination loses the least information for the JTVP.
- **Krylov Subspace Rank Analysis:** This experiment validates whether using the Random Heuristic for selecting JTVP coefficients allows the ns-LSQR algorithm to generate a higher-rank Krylov subspace compared to the Fixed Heuristic.
- **ns-LSQR Heuristic Performance:** This experiment analyzes how the different heuristics and bases affect the accuracy and convergence of the complete ns-LSQR algorithm when solving a linear system. The experiment is run for both well-conditioned and ill-conditioned Jacobians.
- **JVP Method Comparison in ns-LSQR:** This experiment compares the overall performance of the ns-LSQR algorithm when using the Complex-Step method for the JVP versus traditional Forward and Central Difference methods.
- **Newton-nsLSQR Solver for Non-linear Systems:** This experiment evaluates the performance of the full Newton-nsLSQR algorithm on non-linear problems. It analyzes how parameters such as the initial guess and the size of the Krylov subspace affect convergence and efficiency.

4.1. Selected functions and their jacobians

The selection criteria for these functions were influenced by the work of [Moré et al., 1981] and [Sanhueza, 2021].

We define both, the function and its Jacobian for each function.

- **Penalty function II:** It has n unknowns and $m = 2n$ equations. The definition of each

function is given by

$$\begin{aligned} f_1(\mathbf{x}) &= x_1 - 0.2, \\ f_i(\mathbf{x}) &= \sqrt{a}(\exp(x_i/10) + \exp(x_{i-1}/10) - y_i), \quad 2 \leq i \leq n, \\ f_l(\mathbf{x}) &= \sqrt{a}(\exp(x_{l-n+1}/10) - \exp(-1/10)), \quad n < l < 2n, \\ f_{2n}(\mathbf{x}) &= \sum_{j=1}^n (n-j+1)x_j^2 - 1. \end{aligned}$$

where $a = 10^{-5}$ and $y_i = \exp(i/10) + \exp((i-1)/10)$.

- **Penalty Function II Jacobian:** The definition of the Jacobian is given by,

$$J_{pII} = \begin{bmatrix} B \\ C \end{bmatrix}$$

where $B, C \in \mathbb{R}^{n \times n}$ and $c_k = \frac{\sqrt{a}}{10} \exp\left(\frac{x_k}{10}\right)$.

$$B = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 & 0 \\ c_1 & c_2 & 0 & \cdots & 0 & 0 \\ 0 & c_2 & c_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & c_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & c_{n-1} & c_n \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & c_2 & 0 & \cdots & 0 & 0 \\ 0 & 0 & c_3 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & c_{n-1} & 0 \\ 0 & 0 & 0 & \cdots & 0 & c_n \\ 2nx_1 & 2(n-1)x_2 & 2(n-2)x_3 & \cdots & 4x_{n-1} & 2x_n \end{bmatrix}$$

- **Trigonometric function:** It has n unknowns and n equations. For $i = 1 : n$, each function is given by

$$f_i(\mathbf{x}) = n - \sum_{j=1}^n \cos(x_j) + i(1 - \cos(x_i)) - \sin(x_i).$$

- **Trigonometric function Jacobian:**

$$J_{\text{trig}} = \begin{bmatrix} \sin x_1 + 1 \sin x_1 - \cos x_1 & \sin x_2 & \cdots & \sin x_n \\ \sin x_1 & \sin x_2 + 2 \sin x_2 - \cos x_2 & \cdots & \sin x_n \\ \sin x_1 & \sin x_2 & \cdots & \sin x_n \\ \sin x_1 & \sin x_2 & \cdots & \sin x_n \\ \vdots & \vdots & & \vdots \\ \sin x_1 & \sin x_2 & \cdots & \sin x_n + n \sin x_n - \cos x_n \end{bmatrix}$$

- **Extended Powell Singular function:** It has n unknowns and n equations, where $n = 4k$ and $k \in \mathbb{N}$. For $i = 1 : k$, each function is given by

$$\begin{aligned} f_{4i-3}(\mathbf{x}) &= \mathbf{x}_{4i-3} + 10\mathbf{x}_{4i-2} \\ f_{4i-2}(\mathbf{x}) &= \sqrt{5}(\mathbf{x}_{4i-1} - \mathbf{x}_{4i}) \\ f_{4i-1}(\mathbf{x}) &= (\mathbf{x}_{4i-2} - 2\mathbf{x}_{4i-1})^2 \\ f_{4i}(\mathbf{x}) &= \sqrt{10}(\mathbf{x}_{4i-3} - \mathbf{x}_{4i})^2 \end{aligned}$$

- **Extended Powell Singular function Jacobian:**

$$J_{\text{eps}} = \begin{bmatrix} A(1) & 0 & \dots & 0 \\ 0 & A(2) & \ddots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \dots & A(N) \end{bmatrix}$$

where $J \in \mathbb{R}^{n \times n}$ and $A \in \mathbb{R}^{4 \times 4}$ with $N = n \pmod{4}$

$$A(i) = \begin{bmatrix} 1 & 10 & 0 & 0 \\ 0 & 0 & \sqrt{5} & -\sqrt{5} \\ 0 & 2(\mathbf{x}_{4i-2} - 2\mathbf{x}_{4i-1}) & 4(\mathbf{x}_{4i-2} - 2\mathbf{x}_{4i-1}) & 0 \\ 2\sqrt{10}(\mathbf{x}_{4i-3} - \mathbf{x}_{4i}) & 0 & 0 & -2\sqrt{10}(\mathbf{x}_{4i-3} - \mathbf{x}_{4i}) \end{bmatrix}$$

For all experiments, the size of n is 512.

4.2. Basis Comparison for the JTVP

In Section 3.3 we stated that one of the reasons why it is convenient to use a wavelet basis instead of the canonical one is that most of the information of the vector $J^T \mathbf{v}$ can be concentrated in just a few coefficients. This means that less information is lost in the discarded coefficients, in distinction to the canonical basis, where generally the information of a vector is distributed across all coefficients. The objective of this experiment is to verify whether less information is lost from the vector $J^T \mathbf{v}$ when using a wavelet basis.

For this we calculate the relative error of the vector $J^T \mathbf{v}$ using different percentages of coefficients for the approximation. We computed this approximation 30 times for each basis and each percentage using the Random Heuristic (since it is stochastic) and just one time for the other two heuristic (since they're deterministic). The relative error is defined as

$$\text{Relative Error of the JTVP} = \frac{\|J^T(\mathbf{x}_{\text{eval}})\mathbf{v} - \text{JTVP}(\mathbf{x}_{\text{eval}}, \mathbf{v})\|_2}{\|J^T(\mathbf{x}_{\text{eval}})\mathbf{v}\|_2},$$

where $J(x_{eval})^T v$ is the exact product, $JTVP(x_{eval}, v)$ the approximation made by the algorithm [5], x_{eval} a known vector and the point of evaluation of the Jacobian (of the function of interest), and v a known vector.

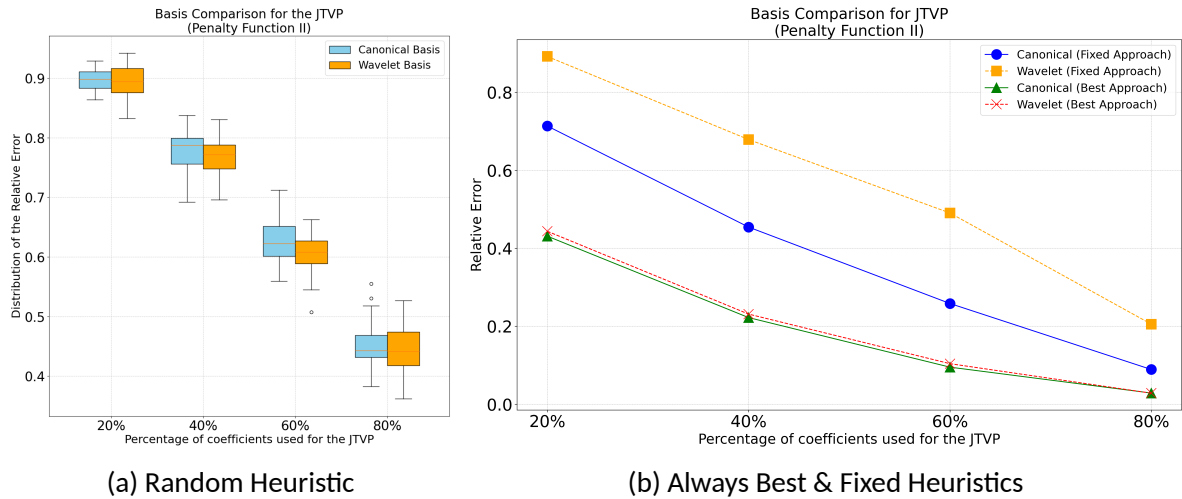


Figura 1: Relative error of the JTVP for the Penalty Function II using both basis and the three heuristics

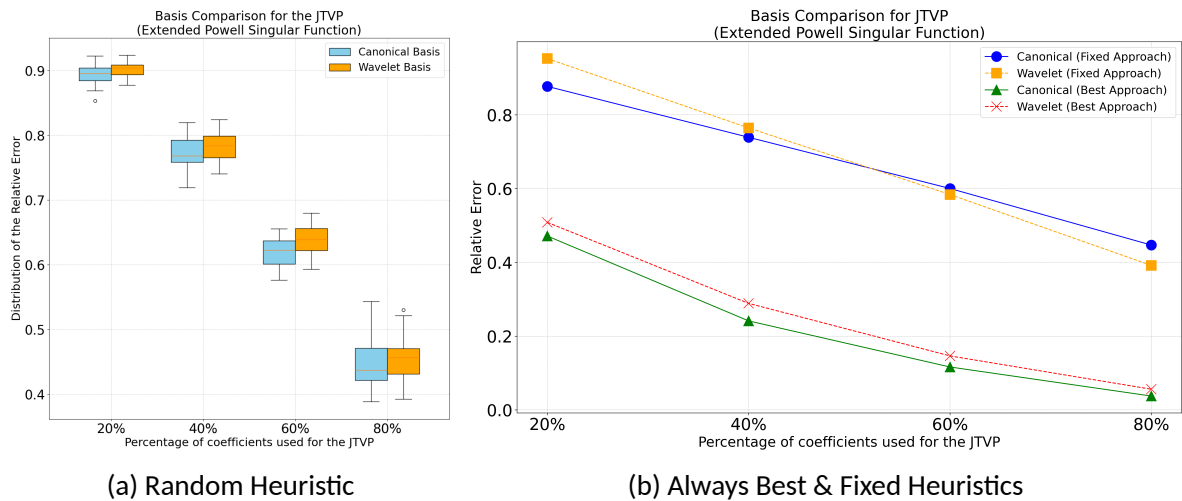


Figura 2: Relative error of the JTVP for the Extended Powell Singular Function using both basis and the three heuristics

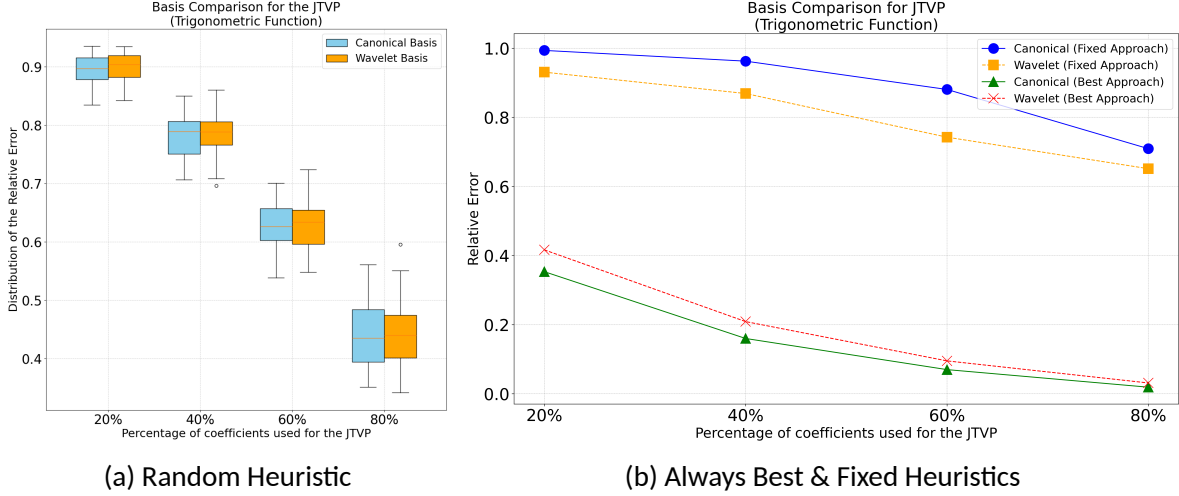


Figure 3: Relative error of the JTVP for the Trigonometric Function using both basis and the three heuristics

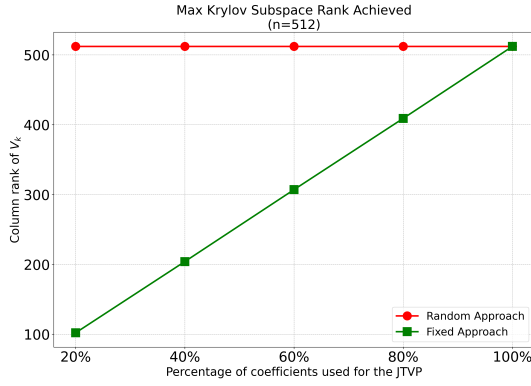
Regarding the information loss of the vector $J^T v$ when using the random heuristic, there are certain occasions where a wavelet basis produces a lower relative error, such as in the Penalty Function II when using a coefficient percentage of 60 %. For the remaining functions, there does not appear to be a significant difference between the wavelet and canonical bases.

With respect to the fixed heuristic, note that for the Penalty Function II and the Extended Powell Singular Function, the errors from this heuristic and the random heuristic are similar. In contrast, for the Trigonometric Function, the random heuristic shows a clear advantage.

As expected, selecting the best coefficients yields the best results in all cases. A key observation is that a coefficient percentage of 20 % with this heuristic performs similar to a coefficient percentage of 80 % with the random heuristic.

4.3. Krylov subspace

The purpose of this experiment is to validate whether the random calculation of the l coefficients of the JTVP, instead of calculating the first l coefficients, can increase the rank of the Krylov subspace in the ns-LSQR algorithm. The experiment consists of running the ns-LSQR algorithm until it stops because the new vector v_k added to V_k is linearly dependent. The rank of V_k is calculated by applying the Singular Value Decomposition (SDV) to the matrix V_k returned by the nsLSQR algorithm and counting the number of singular values greater than a specific tolerance (10^{-15}).



Percentage	F. approach	R. approach
20 %	102	512
40 %	204	512
60 %	307	512
80 %	409	512
100 %	512	512

Figure 4: Max column rank of V_k achieved by the ns-LSQR algorithm

Figure 4 provides a more detailed view of the problem described in Section 3.1. The rank of the Krylov subspace is limited by the same number l of coefficients used to approximate the vector $J^T v$. This, as expected, is due to the way V_k is constructed. However, what is interesting is that the use of random coefficients allows the rank of V_k to reach its maximum (e.i., it is equal to the size of the problem), even when computing only 20 % of the coefficients. This indicates that, indeed, new search directions are being introduced in each iteration using random coefficients. This behavior is equal for the wavelet and canonical bases and all three functions F .

4.4. ns-LSQR heuristic comparison

The purpose of this experiment is to analyze how the heuristics affect the accuracy of the overall ns-LSQR method. Our intention here is to get a better insight into how the coefficient percentage used for approximating the vector $J^T v$ affects the accuracy of the solution given by the ns-LSQR method. Each heuristic will be evaluated using both, the canonical and wavelet basis.

The problem that will be solved by the ns-LSQR method is defined as

$$\min_{\Delta x} g(\Delta x) = \|\mathbf{b} - J(\mathbf{x}_{eval})\Delta x\|_2 \quad (15)$$

where J is the Jacobian matrix (of the function of interest) evaluated at a normally distributed random vector $\mathbf{x}_{eval}$, $\mathbf{b}$ is defined as $J(\mathbf{x}_{eval})\mathbf{y}$, $\mathbf{y}$ is a known vector and the exact solution of the problem, $\mathbf{x}_0$ is the initial guess defined as $\mathbf{x}_0 = \mathbf{y} + 0,001\mathbf{p}$, and $\mathbf{p}$ is a random unitary vector.

For each $\Delta \mathbf{x}_i$ computed at each iteration, the relative error is defined as

$$\text{Relative Error} = \frac{\|(\mathbf{x}_0 + \Delta \mathbf{x}_i) - \mathbf{y}\|_2}{\|\mathbf{y}\|_2}.$$

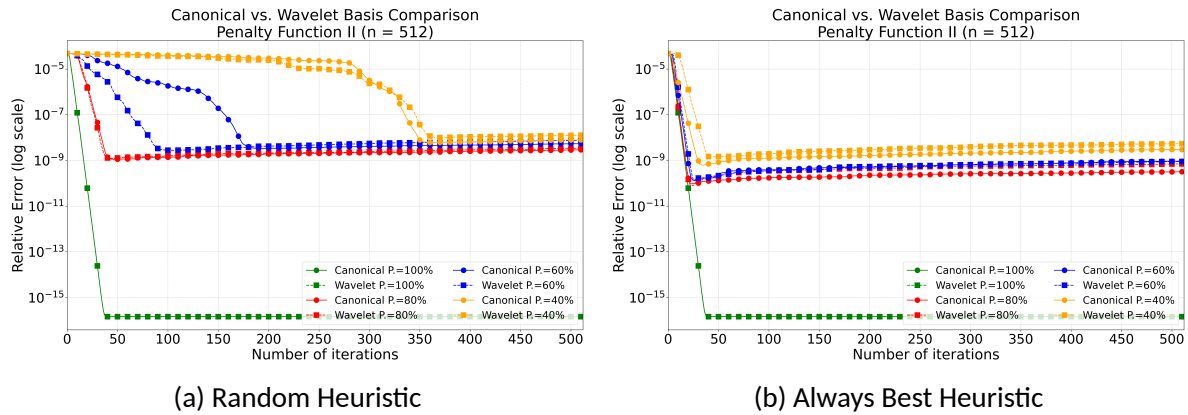


Figure 5: Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm.

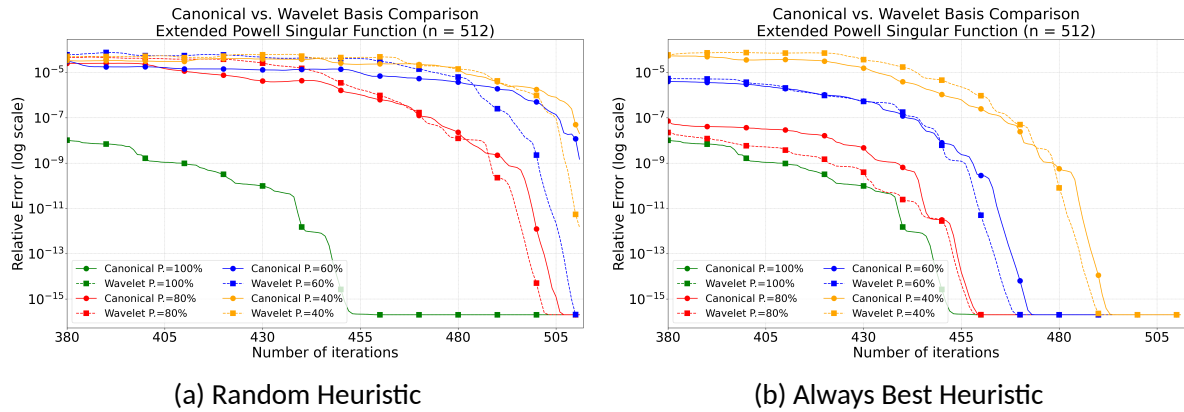


Figure 6: Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm.

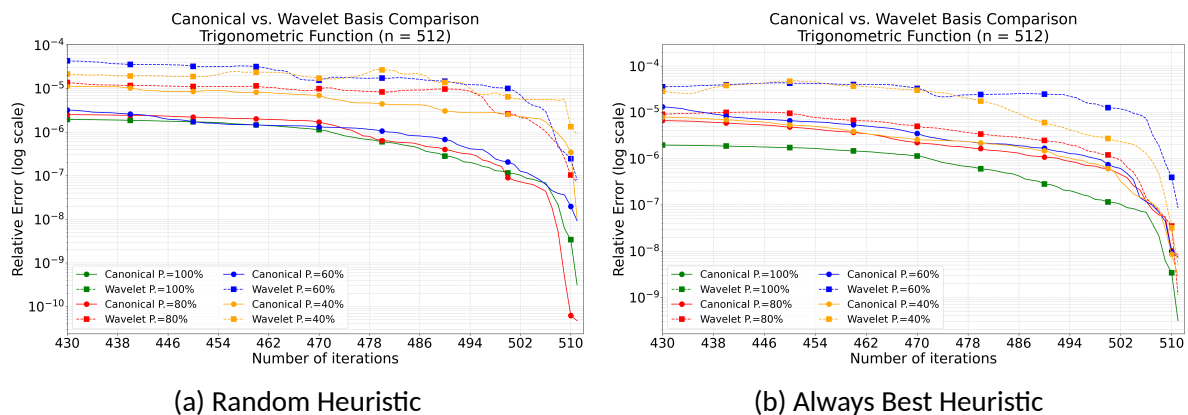


Figure 7: Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm.

In the case of the Penalty Function II, figure 5, an acceleration in convergence can be observed when approximating using only 60 % of its coefficients. A relative error of 10^{-9} is reached at iteration 100 using a wavelet basis, whereas the same relative error is achieved at iteration 170 using the canonical basis. This can be extremely beneficial if considering the use of restart in the nsLSQR algorithm. For other coefficient percentages, there is no significant difference between the two basis.

In the case of the Ext. Powell Singular Function, figure 6, the wavelet basis enables convergence to a smaller final error. Using a coefficient percentage of 60 %, the wavelet basis reaches the exact solution, whereas when using the canonical basis, the minimum error achieved is 10^{-8} . A similar result is observed using a coefficient percentage of 40 %.

Finally, for the Trigonometric Function, figure 7, it is observed that using a wavelet basis is, in fact, detrimental. The use of a canonical basis shows more stability and allows percentages lower than 100 % to converge to smaller errors.

Regarding the efficiency of the Random Heuristic for coefficient selection, a significant gap for improvement exists when compared to the Always Best Heuristic, as seen across all three cases. Future work could involve developing better heuristics. Likewise, using other bases and/or transforms could also significantly improve accuracy and convergence.

In conclusion, these experiments show that there exist situations where the wavelet basis outperforms the canonical basis, nevertheless, in some cases, using a wavelet basis can be detrimental.

4.4.1. The fixed-heuristic problem

In Section 3.3, we saw that the relative error of the JTVP approximation using the fixed-heuristic was quite reasonable, being similar than the random heuristic. However, as established in Section 4.3, a key consequence of this approach is a restriction on the reachable rank of the matrix V_k . We may ask if the lower relative error associated with the fixed-heuristic could enable the nsLSQR algorithm to produce better results, even with a constrained matrix V_k . To investigate this possibility, we re-examine the problem from the preceding section, now implementing the fixed-heuristic.

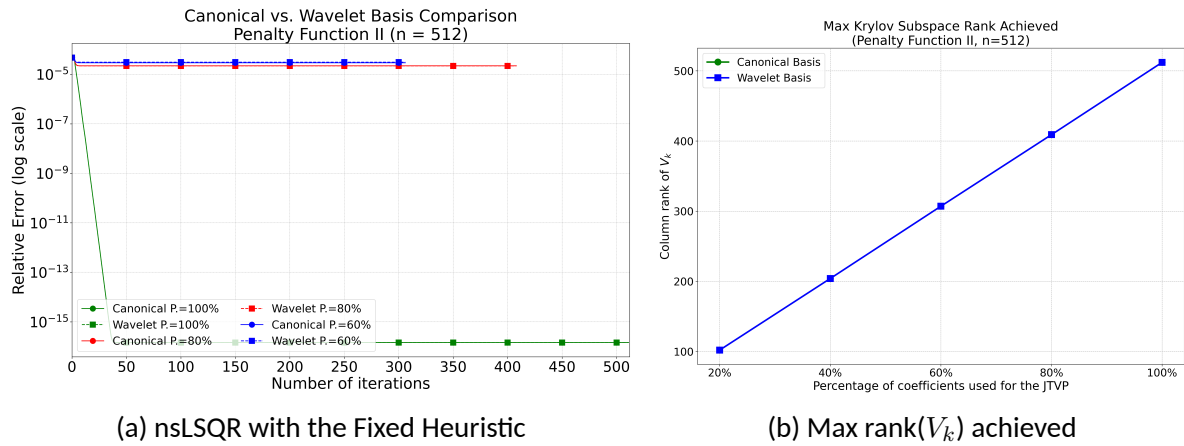


Figura 8: Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.

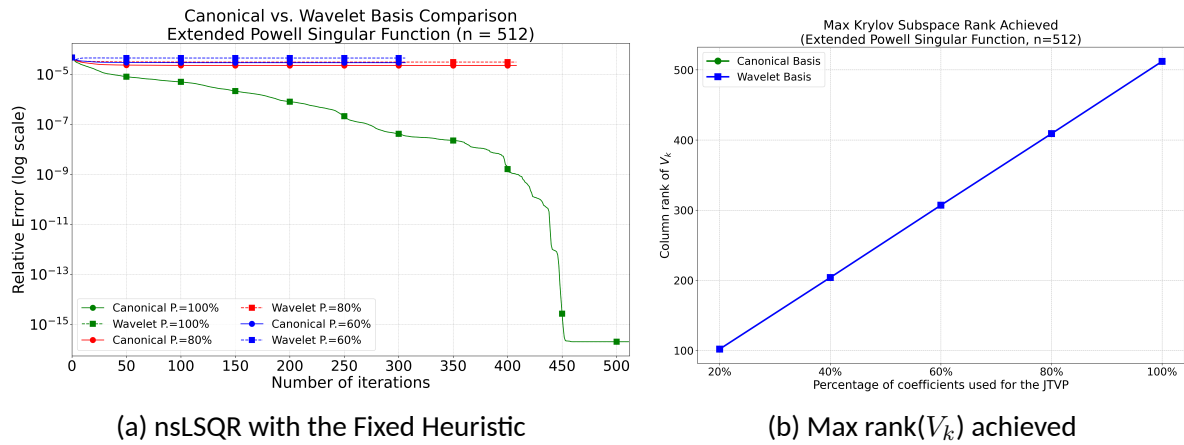


Figura 9: Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.

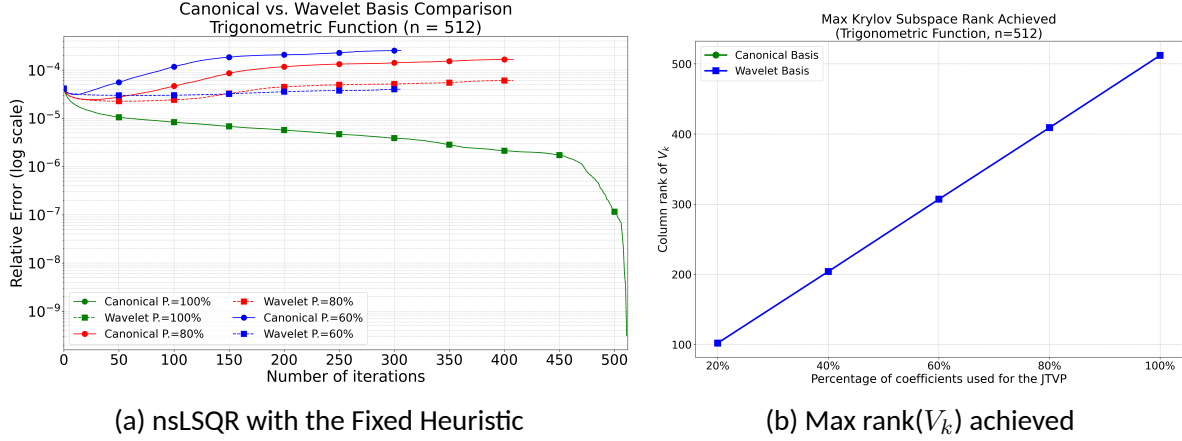


Figura 10: Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm using the Fixed Heuristic and the Max Krylov Subspace rank achieved.

It is observed that in no case is it possible to use a coefficient percentage below 100 %, being clear that a JTVP approximation based on calculating the first l coefficients is unfeasible within the nsLSQR algorithm.

4.4.2. Ill-conditioned jacobian

In the preceding experiments, the evaluation point $\mathbf{x}_{eval}$ was selected to ensure the resulting Jacobian matrix was well-conditioned. This was done to establish a baseline for performance under ideal conditions. Now we investigate the case when the $\mathbf{x}_{eval}$ makes the Jacobian matrix ill-conditioned.

The experimental setup remains the same as in section 4.4, only modifying the vector $\mathbf{x}_{eval}$ in order to make the Jacobian matrix ill-conditioned.

Previously, for each function, the experiments were made with a well-conditioned $\mathbf{x}_{eval}$. That is, the Jacobian evaluated at this vector is, in turn, well-conditioned.

Recall that we work indirectly with the Jacobian when making the approximation of $J^T \mathbf{v}$.

Now we present the results of the Random Heuristic for the case of a ill-conditioned Jacobian.

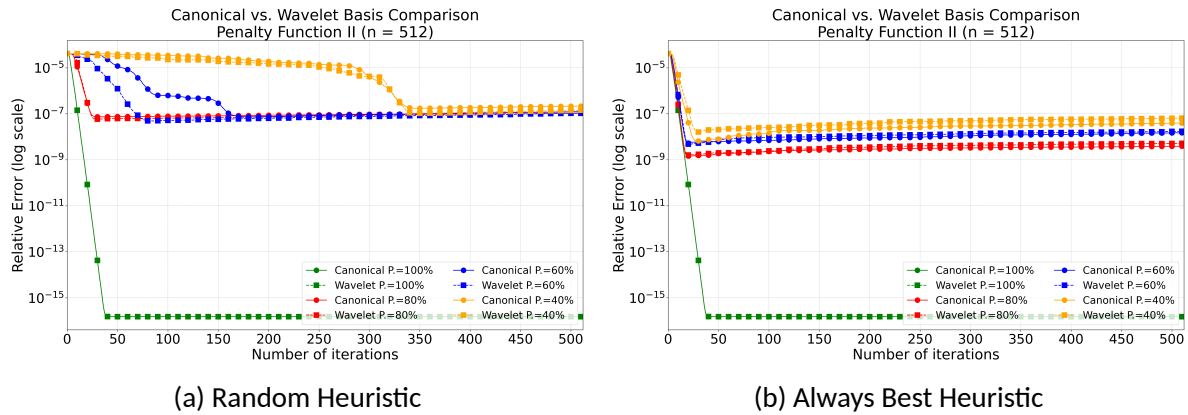


Figure 11: Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm (ill-conditioned case).

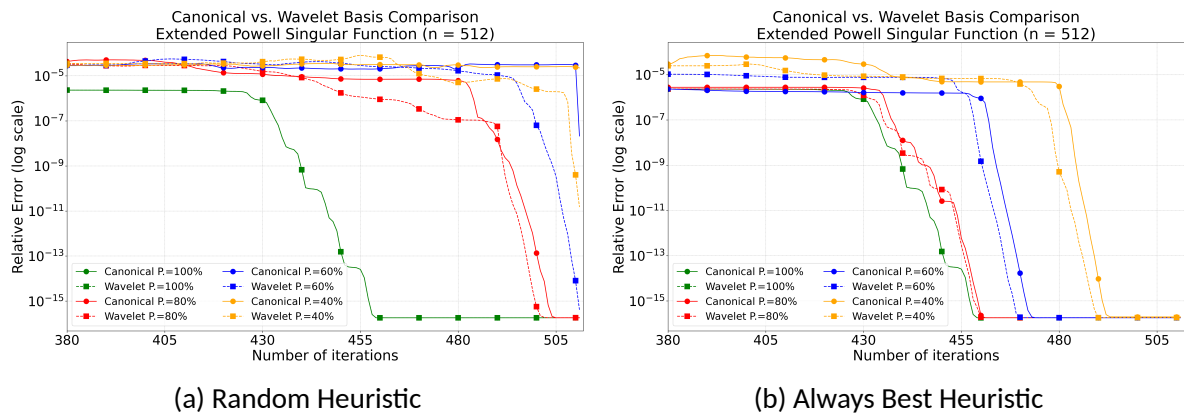


Figure 12: Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm (ill-conditioned case).

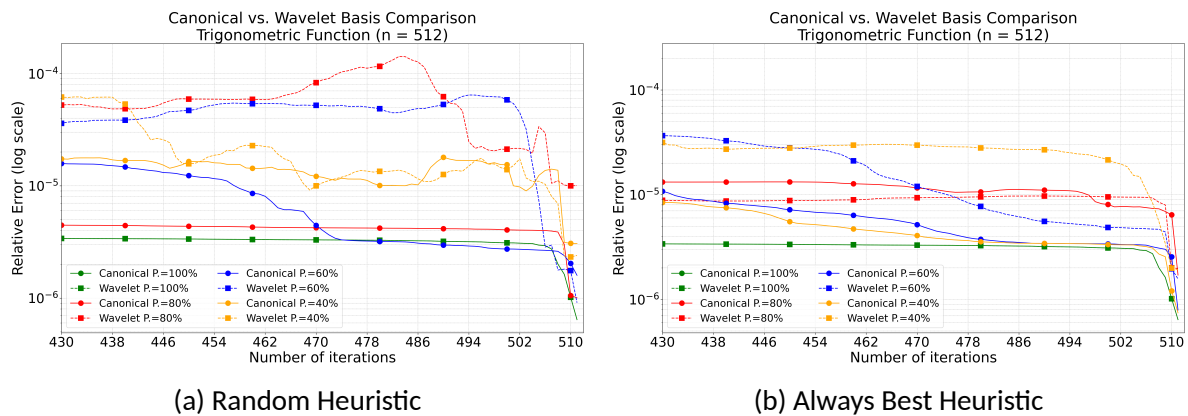


Figure 13: Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm (ill-conditioned case).

The Penalty Function is the only case where the ill-conditioned Jacobian does not interfere with error convergence, the results are very similar to the case of the well-conditioned Jacobian. For the other functions, the ill-conditioned Jacobian significantly degrades the solution's quality, leading to a slower convergence rate and a higher minimum achievable relative error. It is crucial to recognize that this behavior is an inherent characteristic of the functions themselves. This is evidenced by the fact that the solution quality is impacted even when coefficient percentage of 100 % is used for the JTV approximation. One way to mitigate this problem could be to introduce a regularization parameter λ .

4.5. Finite differences vs complex-step in nsLSQR

As discussed in Chapter 3, we said that one of the main reasons for using the Complex-Step method is its superior accuracy in calculating the vector Jw , and its stability with respect to the step size h , especially when compared to other finite difference methods.

To verify this, we now solve the same problem from the previous experiment, using the Forward Difference and Central Difference methods and comparing them with the Complex-Step method. For this analysis, each approximation method used its optimal step size h , which was determined by identifying the value that minimized the relative error in the approximation of the vector $J^T v$, calculating all its coefficients (see figures to the right).

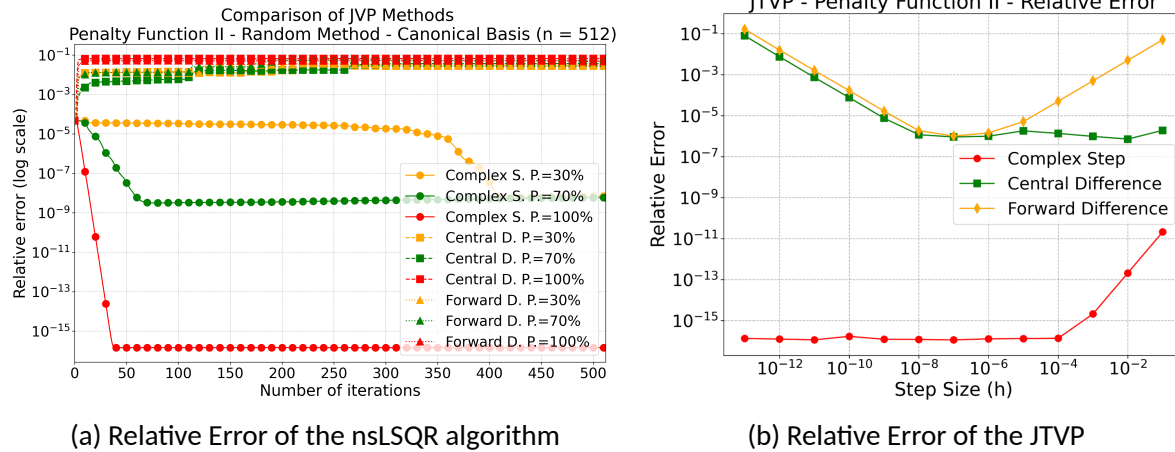
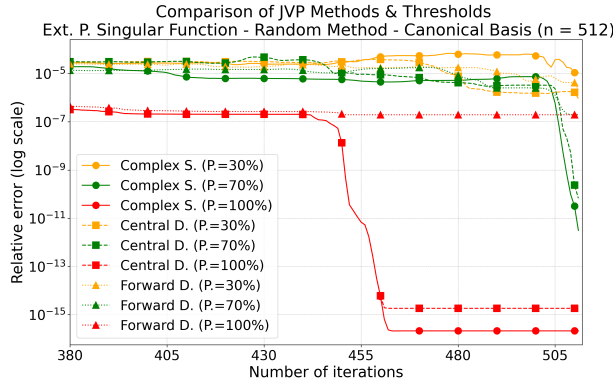
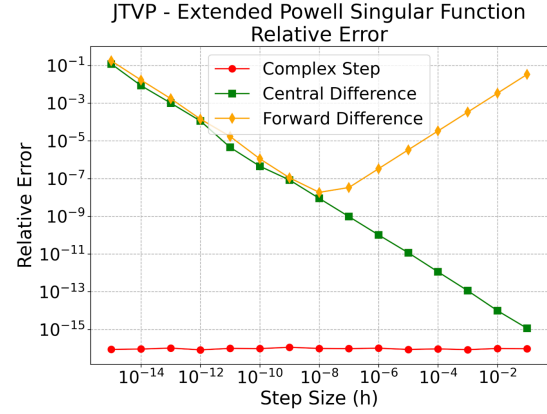


Figura 14: Relative error of the Penalty Function II at each iteration of the nsLSQR algorithm using different finite differences methods.

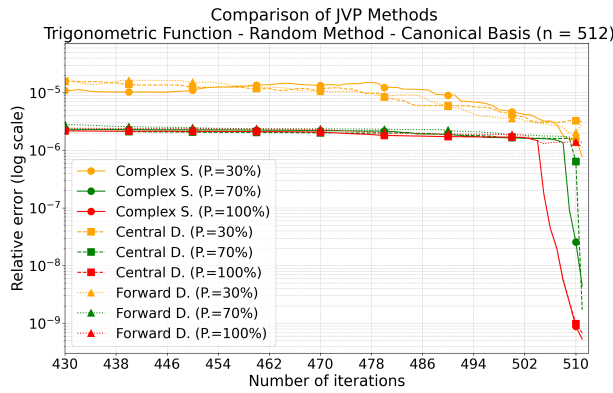


(a) Relative Error of the nsLSQR algorithm

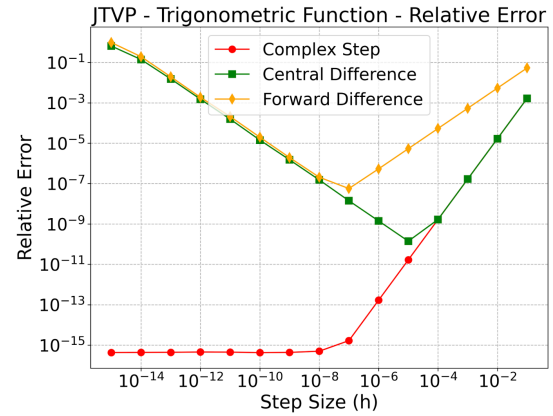


(b) Relative Error of the JTV

Figura 15: Relative error of the Extended Powell Singular Function at each iteration of the nsLSQR algorithm using different finite differences methods.



(a) Relative Error of the nsLSQR algorithm



(b) Relative Error of the JTV

Figura 16: Relative error of the Trigonometric Function at each iteration of the nsLSQR algorithm using different finite differences methods.

For the Penalty Function II, the complex-step method consistently converges to a lower error across all coefficient percentages tested and shows a clear superiority. The results for the Extended Powell Singular Function are more similar. In cases using 70 % and 100 % of the coefficients, the complex step method outperforms the central difference method, but only by a slight margin. Finally, for the Trigonometric Function, both methods yield similar results across all three scenarios.

Overall, these results confirm that the complex-step method is generally the most robust and accurate option. However, it is valuable to note that, under specific conditions, the central difference method can achieve comparable results. Although the complex-step method exhibits markedly superior performance for some functions, this advantage may be less pro-

nounced for others. These latter cases occur when there is no significant difference between the minimum achievable relative error for the JTVF when using either method.

4.6. Newton-nsLSQR comparison

Previously, we examined the performance of the nsLSQR algorithm with the random heuristic for solving over-determined linear problems. To complete this analysis, we now investigate the algorithm's performance on over-determined non-linear problems.

For this, we implement the Newton-nsLSQR algorithm, in which the linear subproblem is solved by the nsLSQR method at each Newton's iteration. Our primary motivation is to determine whether over-determined non-linear problems can be effectively solved with the random heuristic in an efficient way, using the complex-step method.

The problem that will be solved by the Newton-nsLSQR method is defined as

$$\min_x g(x) = \|b - F(x)\|_2,$$

where $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ is the function of interest, b is defined as $F(y)$, y is a known vector and the root of the problem, x_0 is the initial guess and is defined as $x_0 = y + 0.001p$, and p is a random unitary vector. We'll consider that a solution is found when the residual error $g(x_i)$ falls below 10^{-14}

Since our goal is to solve these problems with a reduced memory and computational cost, the problem will initially be solved using a modest Krylov subspace for the nsLSQR algorithm. If convergence is not achieved, the size of V_k will be increased (the choice of the size of V_k is influenced by the results in Section 4.4).

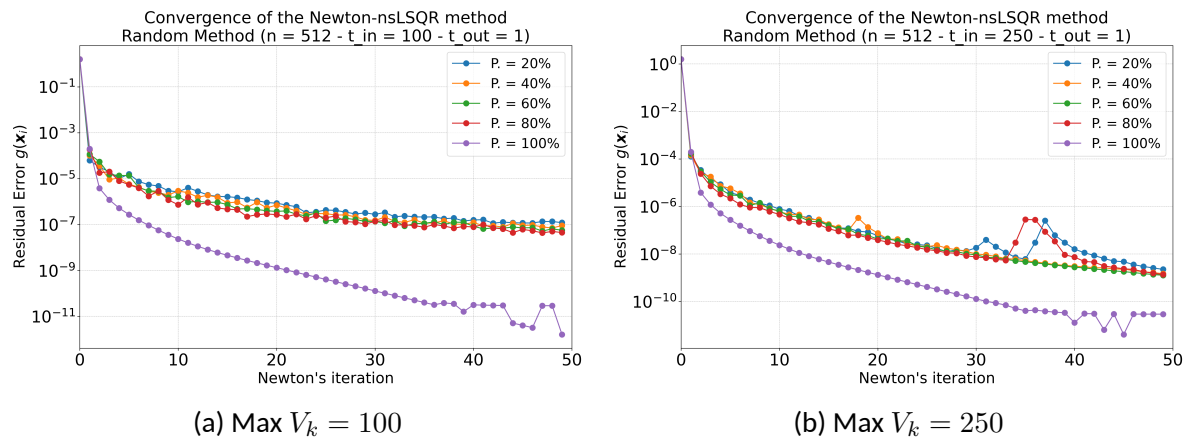


Figura 17: Relative error of the Penalty Function II at each iteration of the Newton-nsLSQR algorithm.

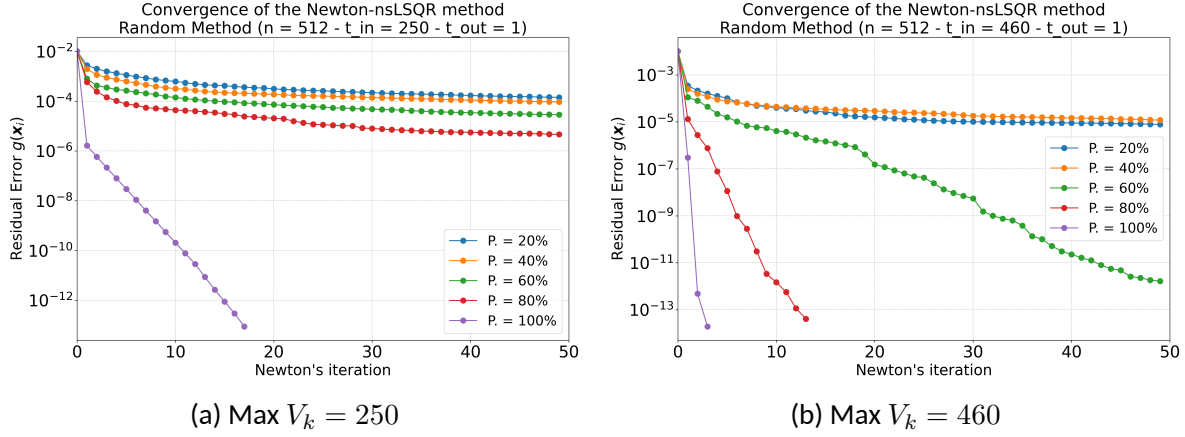


Figure 18: Relative error of the Extended Powell Singular Function at each iteration of the Newton-nsLSQR algorithm.

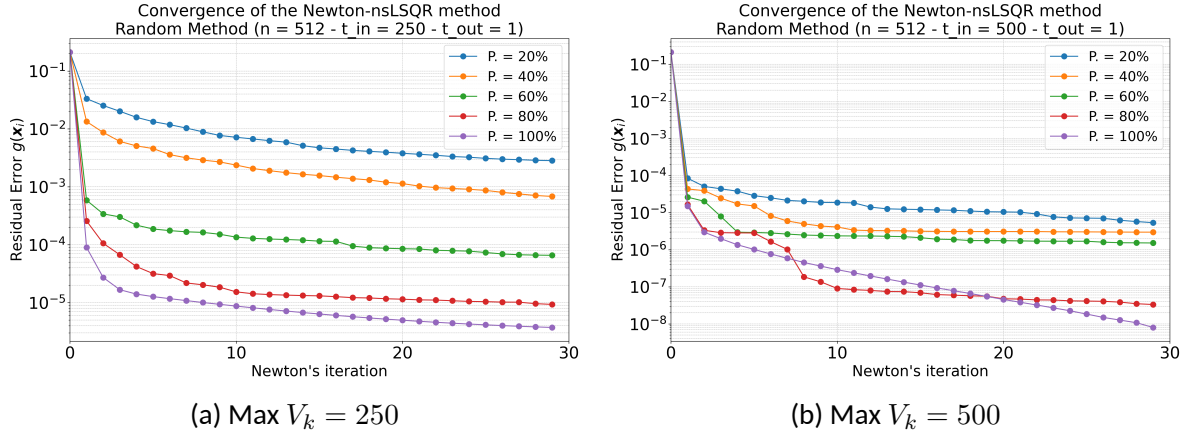


Figure 19: Relative error of the Trigonometric Function at each iteration of the Newton-nsLSQR algorithm.

For the Penalty Function II, there doesn't appear to be a significant impact when a larger Krylov subspace is used. Coefficient Percentages below 100 % require more than 50 iterations to converge to the solution.

For the Extended Powell Singular Function, we can see that if a coefficient percentage lower than 100 % is to be used, it is necessary to increase the Krylov subspace to reach a solution. In any case, it is not advisable to use a coefficient percentage less than or equal to 40 %.

For the trigonometric function, only with a Krylov subspace of $V_k = 500$ (almost equal to the problem size) and a coefficient percentage of 80 % or 100 % a decent convergence rate can be achieved.

In conclusion, solving over-determined non-linear systems of equations in a efficient way is

possible, but the accuracy of the algorithm and the rate of convergence is conditioned by the parameters of the nsLSQR algorithm and the intrinsic characteristics of the system being solved (how the function F is defined).

4.6.1. Initial guess and distance comparison

In addition to the previous experiment, we now analyze how the use of different initial guesses affects the convergence of the Newton-nsLSQR method. It should be noted that, unlike a method such as Levenberg-Marquardt, our implementation is straightforward and lacks any countermeasures (like a regularization parameter) against an ill-conditioned Jacobian or initial guesses that are far from the root of the problem. Therefore, varying the initial guess will give us an insight of the robustness of our implementation. The idea is to use different vectors p and ϵ values for the initial guess $x_0 = y + 0,001p$. A coefficient percentage of 50 % and a $V_k = 100$ is used for the Penalty Function II and a percentage of 70 % and a $V_k = 470$ for both, the Extended Powell Singular and Trigonometric Functions.

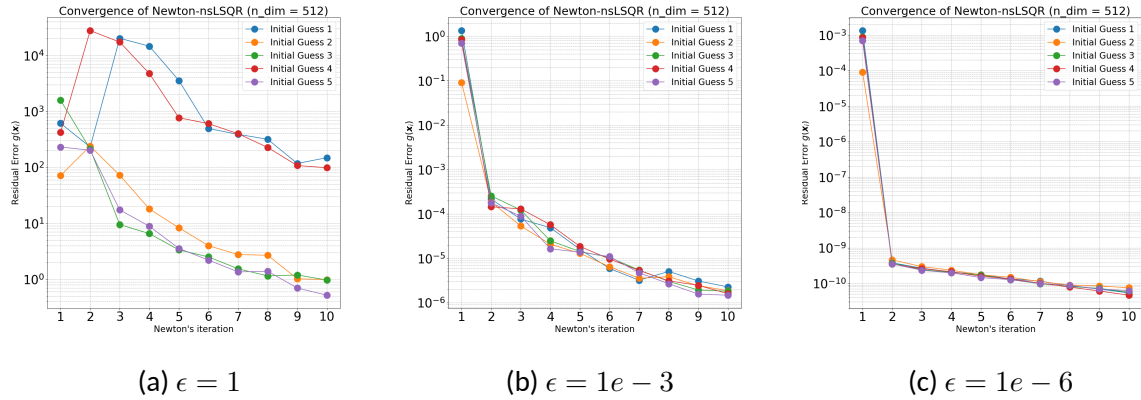


Figura 20: Residual error of the Penalty Function II at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 250$ and a coefficient percentage of 50 % for the JTVP.

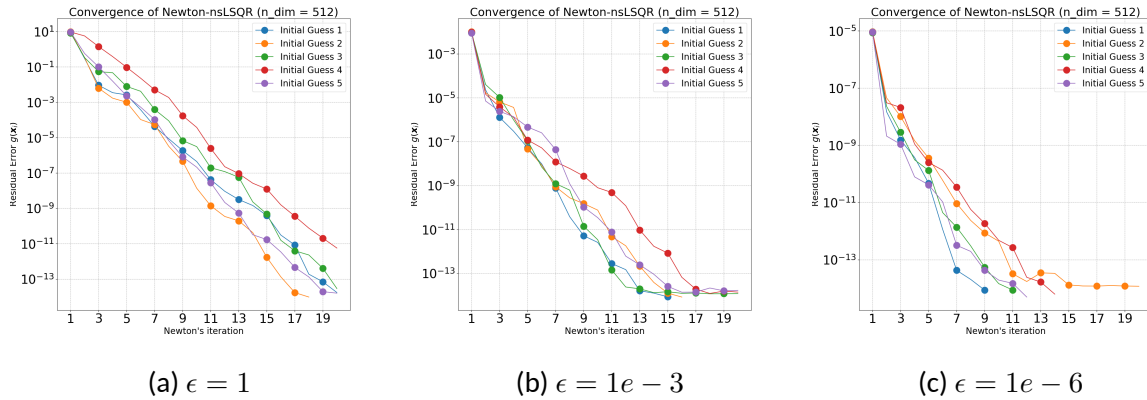


Figure 21: Residual error of the Extended Powell Singular Function at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 470$ and a coefficient percentage of 70 % for the JTVP.

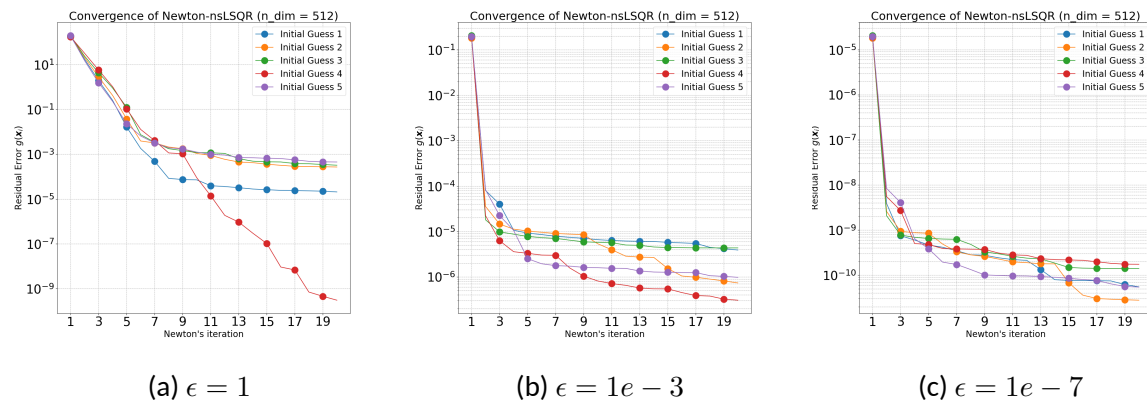


Figure 22: Residual error of the Trigonometric Function at each iteration of the Newton-nsLSQR algorithm using different initial guesses with $V_k = 470$ and a coefficient percentage of 70 % for the JTVP.

The convergence is sensitive to the initial guess. Note that simply changing the initial value (while keeping the same epsilon) affects the convergence. As expected, a closer initial guess leads to a faster convergence rate. However, this also depends on the function being analyzed. The Penalty Function II and the Trigonometric Function show greater sensitivity to the distance of the initial guess, whereas the Extended Powell Singular Function is highly robust, as it finds the solution from all three distances.

Chapter 5

Conclusions

In this investigation, we addressed the significant computational challenge of solving large-scale, over-determined non-linear systems of equations. The primary bottleneck in many iterative methods is the prohibitive cost of computing, storing, and manipulating the full Jacobian matrix. Our work focused on developing and validating a matrix-free approach using the ns-LSQR algorithm, which relies on efficient and accurate computations of the JVP and JTVP.

The central contribution of this work is the development and validation of a heuristic for approximating the JTVP. We first demonstrated that a naive approach, such as using a fixed subset of the first l canonical vectors, is fundamentally flawed. This method confines the solution search to a low-rank Krylov subspace, preventing the ns-LSQR algorithm from converging to the correct solution.

To overcome this critical issue, we proposed the Random Heuristic, which involves approximating the JTVP using a randomly selected subset of basis vectors at each iteration. Our experiments conclusively show that this simple yet powerful strategy is highly effective. By dynamically changing the projection subspace, the Random Heuristic successfully builds a full-rank Krylov subspace, enabling the ns-LSQR algorithm to converge reliably even when using only a small fraction of the basis vectors. This approach, particularly with the canonical basis, provides a robust, general-purpose solution for approximating the JTVP in large-scale contexts.

Furthermore, we explored a refinement to this heuristic by replacing the canonical basis with a Wavelet basis, under the hypothesis that it could offer a sparser representation of the JTVP. The results were promising but problem-dependent. For certain functions, the wavelet basis led to faster convergence, suggesting that when the JTVP is compressible in the wavelet domain, this method can improve computational efficiency. However, for other functions, it offered no advantage or was even detrimental. This indicates that while the wavelet basis is a valuable tool, its effectiveness is not universal and depends on the specific structure of the problem.

Regarding the use of Complex-Step method, the numerical experiments demonstrated that this technique provides significantly higher accuracy and stability across a wide range of step sizes for computing the JVP and JTVP compared to traditional finite difference methods, effectively eliminating step-size selection as a major source of error. While the Complex-Step method exhibits markedly superior performance for some functions, this advantage may be less pronounced for others. Nevertheless, it generally stands as the most stable and accurate option overall.

Finally, by benchmarking against the Always Best heuristic, we confirmed that while our Ran-

dom Heuristic is effective, a significant gap remains between a random selection and an optimal one. This highlights a clear path for future research. Future work could focus on developing more sophisticated, adaptive heuristics that can identify and select more significant coefficients at a lower computational cost than the Always Best Heuristic. Additionally, exploring other orthonormal bases or transforms beyond wavelets could yield further improvements in accuracy and convergence speed for a wider range of problems.

In summary, this work introduces new alternatives to approximate the JVP and the JTV within the ns-LSQR algorithm. The results successfully show that coupling the highly accurate Complex-Step method for the JVP with the proposed Random Heuristic for the JTV provides an effective solution. This combined approach achieves good and efficient results, making it a viable tool for applications where the Jacobian is computationally intractable.

Chapter 6

Declaración de uso de herramientas y tecnologías de Inteligencia Artificial

- Reconozco el uso de Writefull para corregir erratas de la memoria. Es automático, no hace uso de prompts.
- Reconozco el uso de Gemini para traducir del español al inglés ideas y conceptos que se me hicieron difícil de explicar en esa lengua. Usé como prompt 'Traduce lo siguiente al inglés ...'. Usé el output para tener una idea de cómo redactar algunas partes de la memoria.
- Reconozco el uso de Gemini para buscar información relacionada a tópicos de la memoria misma. Usé como prompt 'Busca información con Deep Search sobre este tema (e.g., Transformadas Wavelet, Métodos Iterativos, nsLSQR)'. Usé el output para la complementar la bibliografía de la memoria.
- Reconozco el uso de Gemini para estudiar tópicos de la memoria misma. Usé como prompt 'Explícame esto (e.g., GMRES, Sistemas Sobre-Determinados, Transformada Wavelet)'. Usé el output para hacer un estudio complementario de los tópicos de la memoria.
- Reconozco el uso de Gemini para constantemente corregir mi código. Usé como prompt 'Tengo el siguiente error ... ¿a qué se debe?', '¿Cómo hago que este gráfico sea más grande?'. Usé el output para la corrección y elaboración de gráficos y la implementación correcta implementación de código.

REFERENCIAS BIBLIOGRÁFICAS

- [Armijo, 1966] Armijo, L. (1966). Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3.
- [Birgin et al., 2003] Birgin, E. G., Krejić, N., y Martínez, J. M. (2003). Globally convergent inexact quasi-newton methods for solving nonlinear systems. *Numerical algorithms*.
- [Burrus et al., 1998] Burrus, C. S., Gopinath, R. A., y Guo, H. (1998). Introduction to wavelets and wavelet transforms: A primer. *Prentice Hall*.
- [Castro, 2025] Castro, N. A. (2025). Análisis del método de derivación de paso complejo como alternativa de mitigación de errores de cancelación en la resolución de sistemas de ecuaciones no lineales. *Memoria para optar al título de ingeniero civil en informática*.
- [Curtis, 2019] Curtis, H. D. (2019). Orbital mechanics for engineering students (4th ed.). *Butterworth-Heinemann*.
- [Dennis y Schnabel, 1996] Dennis, Jr., J. E. y Schnabel, R. B. (1996). Numerical methods for unconstrained optimization and nonlinear equations. *SIAM*.
- [Dennis Jr. y Schnabel, 1996] Dennis Jr., J. E. y Schnabel, R. B. (1996). Globally convergent modifications of newton's method. *SIAM*, 6:111–154.
- [Dong y Jorge, 1989] Dong, C. L. y Jorge, N. (1989). On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1–3):503–528.
- [Golub y Kahan, 1965] Golub, G. y Kahan, W. (1965). Calculating the singular values and pseudo-inverse of a matrix. *Journal of the Society for Industrial and Applied Mathematics, Series B: Numerical Analysis*, 2(2):205–224.
- [Kundu y Cohen, 2010] Kundu, P. K. y Cohen, L. M. (2010). Fluid mechanics (4th ed.). *Academic Press*.
- [Marquardt, 1963] Marquardt, D. W. (1963). An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial and Applied Mathematics*, 11(2):431–441.
- [Martins et al., 2003] Martins, J. R. R. A., Sturdza, P., y Alonso, J. J. (2003). The complex-step derivative approximation. *ACM Transactions on Mathematical Software*, 29(3):245–262.
- [Mas-Colell et al., 1995] Mas-Colell, A., Whinston, M. D., y Green, J. R. (1995). Microeconomic theory. *Oxford University Press*.
- [Moré et al., 1981] Moré, J. J., Garbow, B. S., y Hillstom, K. E. (1981). Testing unconstrained optimization software. *ACM Transactions on Mathematical Software (TOMS)*, 7(1):17–41.

- [Murray, 2002] Murray, J. D. (2002). *Mathematical biology: I. an introduction* (3rd ed.). Springer.
- [Nocedal y Wright, 2006] Nocedal, J. y Wright, S. (2006). *Numerical optimization*. Springer Science and Business Media.
- [Paigey, 1982] Paigey, S. (1982). Lsqr:an algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software (TOMS)*, 8(1):43–71.
- [Sanhueza, 2021] Sanhueza, A. R. (2021). A limited-memory levenberg-marquardt algorithm for solving large-scale nonlinear least-square problems. *Master's thesis, UTFSM*.
- [Sanhueza y Torres, 2017] Sanhueza, A. R. y Torres, C. E. (2017). A matrix-free algorithm based on newton's method for overdetermined nonlinear system of equations. *In 2017 36th International Conference of the Chilean Computer Science Society (SCCC)*.
- [Sauer, 1982] Sauer, T. (1982). Newton's method. *Technical report, Argonne National Lab. IL(USA)*.