Master thesis and internship[BR]- Master's thesis : Forced Response Analysis of a Rotordynamic System with Bearing Clearance Nonlinearity[BR]- Integration internsh

Auteur : Ernotte, Guilain
Promoteur(s) : Salles, Loïc
Faculté : Faculté des Sciences appliquées
Diplôme : Master en ingénieur civil en aérospatiale, à finalité spécialisée en "aerospace engineering"
Année académique : 2024-2025
URI/URL : https://gitlab.com/drti/pyharm; http://hdl.handle.net/2268.2/24839

Forced Response Analysis of a Rotordynamic System With Bearing Clearance Nonlinearity

Improvement of Bifurcation Handling of an Open Source Harmonic Balance Solver

Master thesis,
conducted with the aim of obtaining
the Master of Science's degree in Aerospace Engineering

Author	Guilain ERNOTTE
Jury members	Pr. Loïc SALLES — SUPERVISOR
	Pr. Gaëtan KERSCHEN — PRESIDENT
	Pr. Olivier BRULS — EXAMINER
	Pr. Ghislain RAZE — EXAMINER

University of Liège
School of Engineering
Academic year 2024–2025

Abstract

THIS DOCUMENT presents a master's thesis conducted to obtain the Master of Sciences (MSc) degree in Aerospace Engineering. The work is based on a case study proposed by Safran Tech, a research group within Safran S.A. The initial focus of this study is the forced response analysis of a rotordynamic system, which exhibits nonlinear vibratory behavior due to the presence of a gap between one of the bearings and its support. These gaps are intentionally designed and are typically filled with oil, forming an oil-film bearing. This configuration introduces damping between the bearing and its housing. In the event of abnormal rotor unbalance, the outer ring of the bearing may contact the inner wall of the housing. Therefore, it is crucial to evaluate the system's behavior in this regime to ensure that the dynamic response remains within acceptable limits.

The rotor system is analyzed using pyHarm, an open-source harmonic balance solver developed by Safran Tech. The issue raised by the case study is that this solver is unable to compute the frequency response of the rotor system in certain configurations. Following an investigation of the initial problem, it was found that pyHarm cannot address some potential bifurcation points that may arise in the frequency response, aside from simple fold bifurcations.

Consequently, the primary objective of this thesis is to enhance the bifurcation handling capabilities of pyHarm. To achieve this, the bifurcation detection mechanism within pyHarm has been refactored. Additionally, two numerical techniques have been implemented to facilitate transitions between different branches of the system's frequency response solutions. The first technique involves introducing small perturbations into the system dynamics, while the second technique addresses the algebraic bifurcation equation.

Use of AI — Regarding the use of artificial intelligence, the author has utilized generative AI sparingly as a tool for writing this report, only to find synonyms or reformulate some of his own paragraphs. No AI was employed in the development of the computer code.

Keywords — rotordynamics, nonlinear vibration, harmonic balance, numerical continuation, bifurcation, branching, stability, pyHarm

This page intentionally left blank.

Acknowledgements

First, I would like to thank the jury members, for taking the time to review this master thesis. I particularly thank my supervisor, Mr. Salles, for his involvement in my work. Without his guidance, I would have struggle in my preliminary researches.

I also thank Jason Armand and Quentin Mercier, from Safran Tech. They took the time to clearly explain to me the initial problem. Mr. Mercier followed the evolution of my work throughout the semester. The advices and resources he gave me were really valuable.

The recurring meetings and the relaxed atmosphere that Mr. Salles created in the office were key ingredients to the fulfillment of this thesis. Of course, my office partners also contributed greatly to making the work environment very pleasant. I thank them for their friendliness and unwavering good mood at work.

Lastly, I am grateful to my family, who supported me during these (unexpectedly long) studies; and to Andrea, without whom I would have given up before even completing my bachelor's degree.

This page intentionally left blank.

Contents

This page intentionally left blank.

List of Acronyms

ABE	Algebraic bifurcation equation
AFT	Alternating frequency–time
DFTO	Discrete Fourier transform operator
DOF	Degree of freedom
FFT	Fast Fourier transform
HB	Harmonic balance
NFRC	Nonlinear frequency response curve
PC	Predictor-corrector

This page intentionally left blank.

1 Introduction

THE WORK presented in this thesis is based on a case study proposed by Safran Tech, a research group within Safran S.A.. The objective of this proposal is to implement new features into an open-source harmonic balance solver for analyzing the forced response of rotordynamic systems with gap clearance between a bearing and its support. The specific new features will be determined later, following a thorough identification of the needs arising from the rotordynamic system model provided by Safran Tech.

This chapter outlines the context of the thesis. The first part presents the harmonic balance solver in use, referred to as pyHarm, within the broader context of nonlinear structural dynamics. Subsequently, the second part describes the rotordynamic system model that has been provided. It is important to note that many mathematical and physical concepts are introduced in a preliminary manner here; these technical notions will be explained in greater detail in the following chapter.

1.1 Nonlinear Structural Dynamics

The aerospace industry has historically been a leader in the field of structural dynamics [1]. Before the development of the first aircraft in the 1920s, typical industrial machines and vehicles were generally quite heavy. The first vibration modes occurred at such low frequencies that structural resonance was rarely a concern for these mechanical machines. However, as aircraft structures require lightweight and stiff designs, structural resonance has become an unavoidable issue that must be addressed through appropriate dynamic analyses.

In recent decades, there have been significant advancements in computing capabilities. As a result, considerable efforts have been directed toward the development of numerical methods for addressing linear mechanical vibration problems. Today, determining the linear vibration modes of a structure primarily involves numerically solving an eigenvalue problem. A rather comprehensive treatment of linear mechanical vibrations can be found in, *e.g.*, Geradin and Rixen [1].

In parallel, the development of computational methods in nonlinear dynamics has also garnered significant interest in many fields, among others the aerospace sector. To optimize the design of aircraft engines, it is essential for design teams to have increasingly accurate digital representations to enhance the physical understanding of phenomena and the sizing of components.

In this context, time-domain methods, such as direct time integration and shooting techniques, are often employed. These two methods will be discussed in greater detail in the next chapter. For now, it is important to note that while time-domain methods are effective for systems of reasonable size, they are not well-suited for large-scale systems.

1.1.1 Harmonic Balance Method: State of the Art

To solve nonlinear large-scale dynamic systems, the harmonic balance (HB) method has gained popularity in recent decades. In simple terms, the harmonic balance can be viewed as a reduction method that transforms a set of differential equations into a set of algebraic equations, which are notably easier to solve numerically.

The HB method assumes that the solution of the system is periodic and approximates it using a truncated Fourier series, also known as an *ansatz*. This approach involves considering a limited number of harmonics to represent the solution. By substituting this approximation into the differential equations, a set of nonlinear algebraic equations is generated for each of the selected harmonics, with the unknowns being the sought Fourier coefficients of the ansatz. Methods such as Newton-Raphson procedures can then be employed to solve these algebraic equations. The HB method will be presented more formally in the next chapter.

Harmonic balance is thus a frequency method, with its earliest applications traceable to the 1920s in Appleton and Van Der Pol [24]. In this paper, what can be classified as a single-harmonic balance method was employed to study a simple Duffing oscillator. The extension of the method to multiple harmonics appeared in the literature several decades later, with one of the first instances documented by Urabe [25].

In Chapter 2, the harmonic balance method will be described in the context of nonlinear structural dynamics. The harmonic balance is extensively utilized in this field, and a substantial body of literature has been produced on the subject in recent decades. Even within the more specialized area of rotordynamics, which is the focus of this thesis, a wealth of resources is available. Studies have been conducted on topics such as fretting problems [12, 14, 16] and rotor-stator contact problems [18, 14, 13], with the latter being particularly relevant to this thesis. Contact problems are not exclusive to rotordynamics though. For instance, in Zúñiga et al. [21], the harmonic balance method is used to study the vibrations of steam generator inner tubes subjected to clearance impacts.

However, since harmonic balance is fundamentally a technique that simplifies a system of differential equations as long as the response is periodic, this method finds applications across various engineering and scientific fields. Indeed, a multitude of phenomena occurring in nature can be described, at least approximately, through sets of differential equations. This constitutes numerous potential application areas for the harmonic balance method.

More concretely, harmonic balance has found successful applications in electrical engineering, with its relevance compared to time integration methods discussed in Kundert and Sangiovanni-Vincentelli [26]. The method has also been applied in computational fluid dynamics; for example, Hall, Thomas, and Clark [27] formulates a model of unsteady nonlinear flows in turbomachinery using harmonic balance. The primary limitation of the harmonic balance method is that the phenomenon under study must be periodic.

Finally, readers seeking a broad overview of recent developments and applications of the harmonic balance method can refer to Yan et al. [20].

1.1.2 pyHarm: A Harmonic Balance Method Solver

In this context, a working group at Safran Tech began developing their own harmonic balance solver, which they named pyHarm [28]. An open-source version of this project is available on GitLab. The project is still in the development phase, with hopes of being utilized for practical applications in the near future.

This solver is written in Python and follows a modular approach, where each major solver component is defined by distinct abstract classes. This design allows the codebase to be easily extended with contributions from external developers. The technical implementation details of pyHarm are not discussed in the main chapters of this thesis to maintain conciseness. Readers may refer to the appendices for more technical information on pyHarm. In particular, Appendix A explains how the code is organized in pyHarm and how to adequately contribute new features to the existing codebase. Additionally, Appendix B provides resources for effectively using pyHarm in practice.

As with any solver based on the HB method, the pyHarm software is fully capable of solving linear systems. However, the HB method is particularly advantageous for studying nonlinear systems. As an introductory example to demonstrate the capabilities of pyHarm, it is insightful to focus on the Duffing oscillator and derive some solutions from it. This oscillator is a simple single degree-of-freedom (DOF) nonlinear system, mathematically expressed as:

$$m\ddot{q} + c\dot{q} + kq + \gamma q^3 = f(t)\,. \tag{1.1}$$

In the context of structural mechanics, q represents the displacement of the DOF, while m, c, and k are the mass, damping, and stiffness parameters of the system, respectively. The cubic stiffness parameter γ introduces nonlinearity to the system. Additionally, the system can be excited by an external force f.

The Duffing oscillator frequently appears as an academic example in many textbooks addressing nonlinear dynamics [17, 9, 10]. It serves as a simple yet illustrative example that can exhibit rich and complex dynamical behaviors. The Duffing oscillator has also been the subject of numerous case studies and is often used as a well-known nonlinear system for testing new computational techniques [11, 8, 14]. This will also be the case in this thesis.

Three different frequency responses of the Duffing oscillator are presented in Figure 1.1. These responses are derived from solutions computed by pyHarm, using different values for the Duffing system parameters. An in-depth explanation of the creation of this figure is provided in Appendix B.

The response of the linear system exhibits a simple, straight resonance peak. Due to the relatively strong damping, this peak occurs slightly before the resonance peak of the free system, which is expected to occur at $\sqrt{k/m} = 1$ rad/s.

In contrast, the two nonlinear frequency response curves (NFRC) demonstrate that the system response is no longer a one-to-one relationship with the excitation frequency. This is the first indication of nonlinear behavior: for a given excitation frequency, the system response can exhibit multiple possible states. The state to which the system will eventually stabilize depends on the initial conditions.

As the number of harmonics used in the ansatz increases, a second manifestation of nonlinear behavior becomes evident: higher-order harmonics contribute to the system

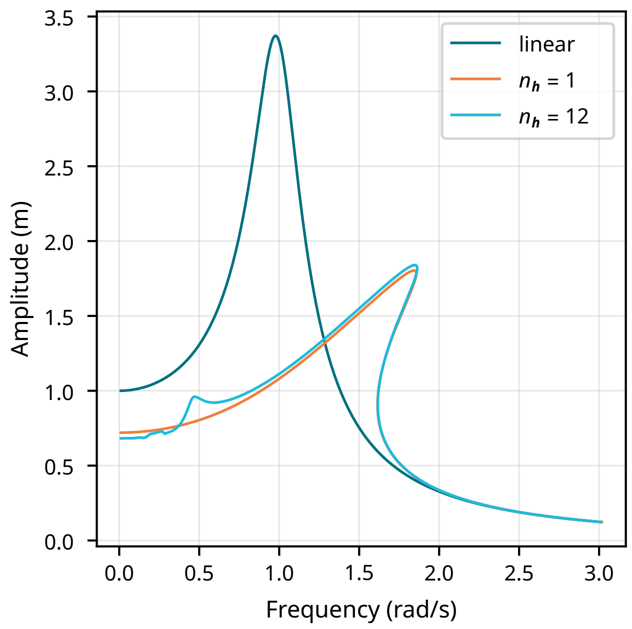


Figure 1.1 Linear and nonlinear frequency responses computed with pyHarm, for the Duffing oscillator of Equation (1.1). Parameters have been set to $m = 1\,\mathrm{kg}$, $c = 0.3\,\mathrm{N\,s/m}$, and $k = 1\,\mathrm{N/m}$. A cubic coefficient of $\gamma = 1\,\mathrm{N/m^3}$ has been chosen for the two nonlinear curves, each computed with 1 and 12 harmonics (n_h). For the linear response, γ is simply set to $0\,\mathrm{N/m^3}$.

response. This behavior is observable in the figure, as indicated by the small frequency peaks occurring at frequencies below 0.5 rad/s.

Overall, the seemingly innocuous introduction of a cubic term gives rise to a range of behaviors not encountered in linear dynamics. These rich behaviors make nonlinear systems much more challenging—and interesting—to study.

It is important to note that Figure 1.1 has only been described qualitatively here. However, deriving the harmonic balance equations analytically for a simple Duffing oscillator with only one harmonic is an excellent way to quickly grasp the working principles of the method. Such a development can be found in, for example, the introductory example of Krack and Gross [9].

1.2 Rotordynamic Systems

As mentioned in the foreword of this chapter, the potential improvements for pyHarm proposed in Chapter 4 will stem from a preliminary study on a specific rotordynamics model. More precisely, engineers at Safran Tech have identified that pyHarm encounters difficulties in computing the frequency response of rotordynamic systems under certain conditions. The issue arises when a gap between one of the shaft bearings and its housing is introduced in the rotor model. In such cases, a sufficiently large unbalance forcing near resonance frequencies may lead to contact between the bearing's outer ring and the inner wall of the housing.

As will be explained in Chapter 3, the manifestation of this bilateral contact results in nonlinear stiff equations that can be challenging to solve numerically if not addressed carefully. However, it is crucial to evaluate the system's behavior in this regime to ensure that the dynamic response remains within acceptable limits.

These gaps between bearings and their housings are intentionally designed and are typically filled with oil, forming a squeeze film damper. Additionally, the outer ring of the bearing can be directly attached to a squirrel cage, which consists of an elastic cage support that provides additional radial stiffness between the bearing and the housing. Squeeze film dampers and elastic cage supports thus offer two critical levers for controlling the vibration levels of the rotordynamic system. A clearer understanding of these concepts can be gained by referring to Figure 1.2.

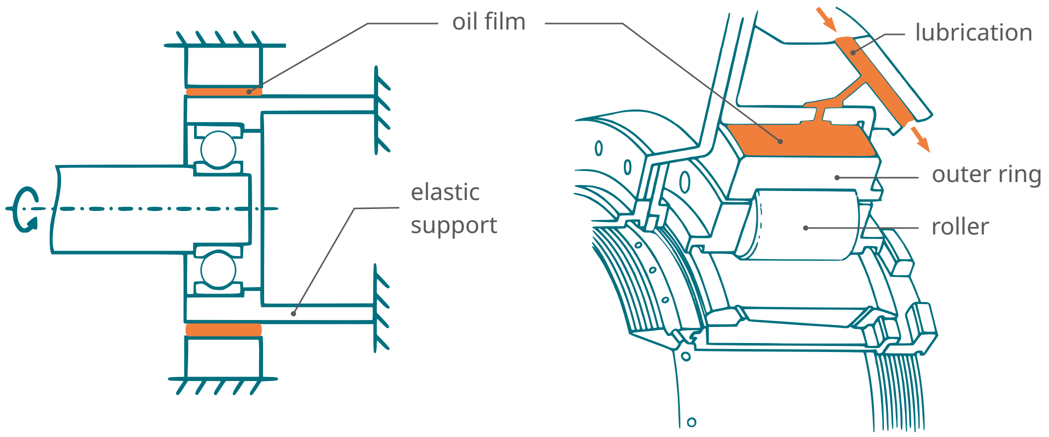


Figure 1.2 Schematic illustrations of squeeze film dampers and elastic squirrel cage support. The figure on the left is adapted from Chen et al. [22] and shows a ball bearing with a squeeze film damper and squirrel-cage support. The figure on the right is adapted from Rolls Royce [23] and depicts the application of lubricant on a bearing's outer ring to form a squeeze film.

As illustrated on the right in Figure 1.2, squeeze films are typically very thin; the left illustration exaggerates the gap size for clarity. Lubricant is retained in the clearance gap either by a close axial clearance in the bearing housing or by a piston ring seal at each end of the film [23].

1.2.1 Safran Toy Rotor Model

The difficulties encountered by pyHarm were observed in a closed-source rotordynamic system. To facilitate the subsequent analysis in this thesis, Safran Tech provided a simplified numerical model of this system. This toy model is sufficiently simple to allow computations to be performed easily on a standard personal computer. A schematic representation of this model is shown in Figure 1.3.

As illustrated in the figure, the downstream bearing, located at node 3-5, is supported by an elastic cage. A squeeze film, situated between nodes 5 and 6, provides damping between the bearing and its housing. As indicated in the figure, the bearing-support interface can be regarded as a mechanical subsystem of the entire rotor structure. This configuration allows for the provision of additional stiffness and damping to the rotordynamic system at the interface, which, as explained in the previous section, aids in controlling rotor vibrations.

However, the bearing-support interface must also account for the clearance gap. This gap is essential for the proper operation of the thin squeeze film. Under normal operating conditions, the squeeze film should effectively dampen any radial movement of the shaft, preventing the bearing from colliding with the inner walls of the housing. In this scenario, the clearance gap does not induce substantial nonlinear behaviors. The analysis becomes more complex when the squeeze film fails to prevent these bearing-housing collisions. A detailed examination of this case will be the focus of Chapter 3.

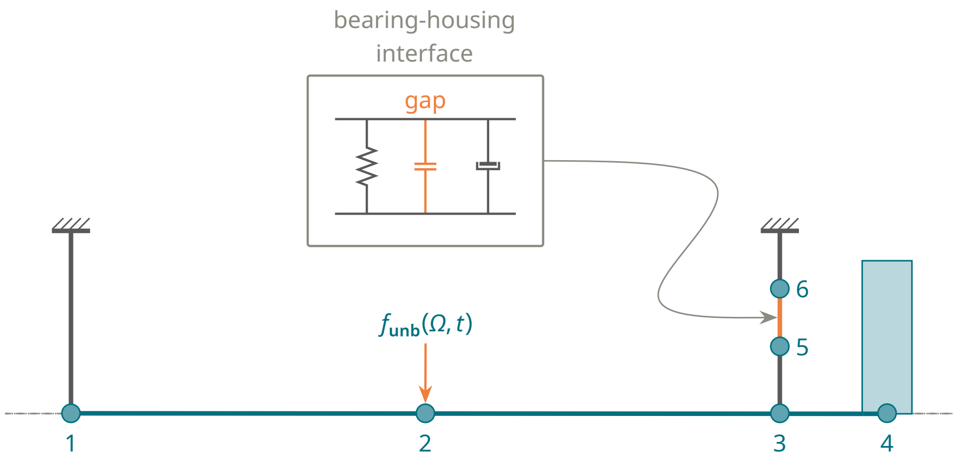


Figure 1.3 Schematic representation of the rotor model provided by Safran Tech. Only the portion of the system above the axis of revolution is depicted. The nodes labeled from 1 to 6 are used to describe the geometric configuration of the system. The rotor's shaft is supported by two bearings, which secure the shaft at nodes 1 and 3. A blade is attached to one end of the shaft at node 4. A potential rotor unbalance is represented by an equivalent excitation force acting radially on the shaft at node 2.

2 Technical Background

PRIOR TO DIVING into the proper analysis of the rotordynamic system under interest, this chapter introduces some of the technical concepts that will appear reccurently throughout the following discussions of this thesis. Only the common theoretical basis is presented in here. Further technical explainations will be introduced gradually when needed.

The chapter begins by outlining the working principles of the harmonic balance method. It is first presented within the broader context of weighted residual techniques. Subsequently, the classical formulation of the HB equations is derived. Following this, the formalization of the HB equations for the analysis of a generic mechanical system is discussed, leading to an investigation of the nonlinear forces. To facilitate this, a companion method to harmonic balance, namely the alternating frequency-time (AFT) domain procedure, will be introduced.

The chapter then explains how the HB equations can be solved numerically for a specific frequency. Afterward, the approach to solving the HB equations across a range of frequencies will be addressed, with a focus on numerical continuation. Emphasis will be placed on the predictor-corrector method.

Finally, the chapter presents elementary definitions related to system stability and bifurcation points. Effective methods for detecting fold bifurcations and branching points will be provided.

As stated in Section 1.1.1, the harmonic balance method is widely used across various fields, and a range of techniques has developed around HB and continuation methods. This chapter primarily focuses on a selected portion of these numerical methods, specifically those that are effectively utilized in pyHarm.

2.1 Harmonic Balance Method

The harmonic balance method have been succinctly summarized in the overview provided in Section 1.1.1: it consists of a reduction method that transforms a set of nonlinear differential equations into a set of nonlinear algebraic equations, referred to as the *HB equations*.

This section presents the main mathematical derivations of the harmonic balance method. It begins by explaining the general principles of a weighted residual method, which serves as a foundation for introducing harmonic balance, a specific case of a weighted residual method. The classical HB equations are then derived. Following this, the application of the HB method to mechanical systems is demonstrated. Additionally, a companion method, the alternating frequency-time procedure, is introduced. This procedure can be used to systematically evaluates the nonlinear forces. Finally, the section discusses alternative techniques for solving nonlinear systems and compares them to the harmonic balance method.

Most mathematical derivations presented in this section are based on the theoretical developments found in Detroux et al. [6] and Krack and Gross [9]. Readers seeking a more comprehensive introduction to the harmonic balance method may refer to the latter reference.

While these two references [6, 9] are not the only sources providing a mathematical derivation of the HB method, it is important to note that there are some divergences in notation and definitions among the various documents addressing harmonic balance. In this section, the conventions and notations used are selected to align closely with those employed internally by pyHarm.

2.1.1 Weighted Residual Methods

Weighted residual methods constitute a family of reduction techniques, of which the harmonic balance method is a part. These methods are utilized to simplify the analysis and derive approximate solutions for periodic boundary value problems. Such a problem can be expressed mathematically in a generic form as follows:

$$\boldsymbol{r}\left(\dot{\boldsymbol{q}},\, \boldsymbol{q},\, t\right) = \boldsymbol{0}\,, \tag{2.1}$$

$$\boldsymbol{q}(t) = \boldsymbol{q}(t+T)\,. \tag{2.2}$$

Equation (2.1) represents a system of differential equations that models the dynamics of the problem under investigation. Here, $\boldsymbol{q} \in \mathbb{R}^{n_q}$ denotes the vector of dependent variables, and $t \in \mathbb{R}$ is the sole independent variable of the problem. The vector $\dot{\boldsymbol{y}} \in \mathbb{R}^{n_q}$ indicates the derivative of $\boldsymbol{q}$ with respect to the independent variable. Finally, $\boldsymbol{r}$ is a function that maps $\mathbb{R}^{2n_q+1}$ to $\mathbb{R}^{n_q}$.

Equation (2.2) imposes a periodic boundary condition on the system of differential equations, indicating that the solution $\boldsymbol{q}$ must repeat its values at regular intervals of T. In this context, $\boldsymbol{q}$ is referred to as *periodic*.

To approximate the solution of Equation (2.1), a weighted residual method first proposes to express the solution $\boldsymbol{q}$ as a linear combination:

$$\boldsymbol{q}(t) \approx \boldsymbol{q}_{\mathrm{h}}(t,\, \{\boldsymbol{\beta}_k\}) = \sum_{k=1}^{B} \boldsymbol{\beta}_k\, b_k(t)\,. \tag{2.3}$$

This approximation is referred to as an *ansatz*. From a linear algebra perspective, this approach involves projecting the solution $\boldsymbol{q}$ onto a selected set of basis functions $b_k(t)$. This represents a simplification of the original problem, as the goal now is to determine the values of the coefficients $\boldsymbol{\beta}_k \in \mathbb{R}^{n_q}$ that will make the approximation $\boldsymbol{q}_{\mathrm{h}}$ as close as possible to the true solution $\boldsymbol{q}$. To estimate these coefficients $\boldsymbol{\beta}_k$, it is useful to first examine the implications of introducing the ansatz into the system's dynamics.

Equation (2.1) indicates that the system is balanced when the mapping $\boldsymbol{r}$ yields $\boldsymbol{0}$. In this case, $\boldsymbol{q}$ represents the solution of the system. However, when the approximate form of the solution $\boldsymbol{q}_{\mathrm{h}}$ is introduced into these equations, Equation (2.1) is generally no longer satisfied. Specifically, the mapping $\boldsymbol{r}$ does not yield $\mathbf{0}$:

$$\boldsymbol{r}_\mathrm{h}(t, \{\boldsymbol{\beta}_k\}) = r(\dot{\boldsymbol{q}}_\mathrm{h}(t, \{\boldsymbol{\beta}_k\}), \boldsymbol{q}_h(t, \{\boldsymbol{\beta}_k\}), t) \neq \mathbf{0}\,. \tag{2.4}$$

In this context, $\boldsymbol{r}_\mathrm{h}$ is referred to as the *residual* of Equation (2.1).

Based on these observations, the weighted residual method proposes a strategy for determining the coefficients $\boldsymbol{\beta}_k$. The central idea of this method is to require that the residual $\boldsymbol{r}_\mathrm{h}$ be orthogonal to a selected set of weight functions $\rho_j(t)$:

$$\frac{1}{T}\int_0^T \rho_j(t)\,\boldsymbol{r}_\mathrm{h}(t,\{\boldsymbol{\beta}_k\})\,dt = \mathbf{0} \quad j = 1, \dots, B\,. \tag{2.5}$$

In other words, this requirement ensures that Equation (2.1) is satisfied in a weighted average sense.

Note that $\boldsymbol{r}_\mathrm{h}(t,\{\boldsymbol{\beta}_k\})$ is a function solely of the independent variable t and the sought coefficients $\boldsymbol{\beta}_k$. Since the weight functions $\rho(t)$ are known, it follows that Equation (2.5) constitutes a system of $n_q \times B$ algebraic equations, where the unknowns are the coefficients $\boldsymbol{\beta}_k$. The dependence on the independent variable indeed vanishes during integration. Consequently, this system of algebraic equations can be solved, allowing for the determination of the ansatz $\boldsymbol{q}_\mathrm{h}$, which serves as an approximation of the true solution $\boldsymbol{q}$.

It is important to note that requiring the residual to be orthogonal to certain weight functions is not equivalent to requiring the residual to be exactly zero. Thus, satisfying the algebraic system in Equation (2.5) is a less stringent requirement than satisfying the dynamics of the original problem.

The difference in requirements imposed by the strong form and the weak form is better illustrated in Figure 2.1. To satisfy Equation (2.1), each component of $\boldsymbol{r}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t)$ must be zero for all values of t. In contrast, for the weak form, Equation (2.5) is satisfied as long as the residuals are orthogonal to the weight functions.

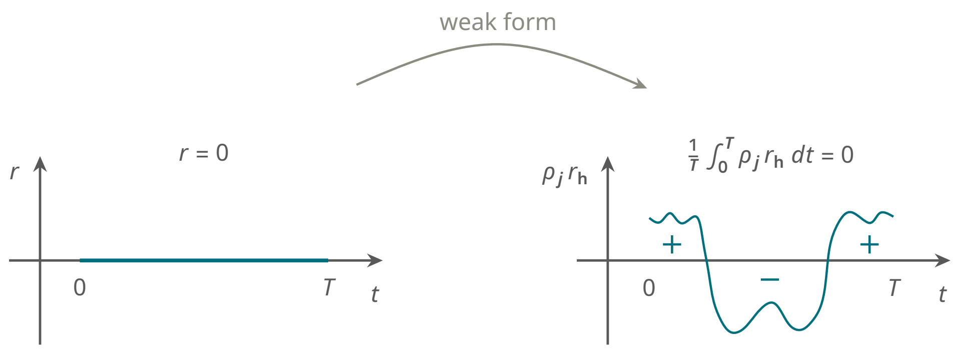


Figure 2.1 Illustration of the conditions that should be satisfied for the original problem of Equation (2.1), and for the weak form provided by the weighted residual method. Drawing reproduced and adapted from Krack and Gross [9].

Under the assumption of periodicity (Equation (2.2)), the weighted residual method reduces a system of differential equations to a system of algebraic equations, accepting a trade-off by relaxing the solution requirements. The original problem and its simplified version are referred to as the *strong form* and the *weak form*, respectively.

It should be emphasized that an ansatz $\boldsymbol{q}_\mathrm{h}$ that perfectly satisfies Equation (2.5), *i.e.*, the weak form, is *not* an exact solution of the original problem. This distinction should always be kept in mind when employing the weighted residual method—and consequently, the harmonic balance method: the coefficients $\boldsymbol{\beta}_k$ computed by a weighted residual solver solely represent a solution to the weak form of the problem.

2.1.2 Classical HB

As previously mentioned, the harmonic balance method is a specific case of a weighted residual method. More specifically, complex exponentials are selected as both the basis functions $b_k(t)$ for the ansatz and the weight functions $\rho_j(t)$ for the weak form requirements.

The harmonic balance method employs a truncated Fourier series to represent the ansatz $\boldsymbol{q}_\mathrm{h}$. Denoting Ω as the circular frequency corresponding to $2\pi/T$, the ansatz can be expressed as follows:

$$\begin{aligned}\boldsymbol{q}_\mathrm{h}(t) &= \sum_{k=-n_h}^{n_h} \hat{\boldsymbol{q}}_k \,\mathrm{e}^{ik\Omega t} \\ &= \boldsymbol{\beta}_1 + \sum_{k=1}^{n_h} \boldsymbol{\beta}_{2k}\cos(k\Omega t) + \boldsymbol{\beta}_{2k+1}\sin(k\Omega t) \\ &= \hat{\boldsymbol{c}}_0^q + \sum_{k=1}^{n_h} \hat{\boldsymbol{c}}_k^q \cos(k\Omega t) + \hat{\boldsymbol{s}}_k^q \sin(k\Omega t)\,.\end{aligned} \tag{2.6}$$

If it is assumed that the the ansatz and the Fourier coefficients $\hat{\boldsymbol{q}}_k$ are real valued, then the truncated Fourier series of the ansatz can be expressed equivalently in terms of complex exponentials or as a combination of sine and cosine functions. The expansion in terms of sine and cosine functions will be retained in the following sections, as it aligns more closely with the internal treatment of computed quantities in pyHarm.

For clarity in the subsequent discussion, the notations $\hat{\boldsymbol{c}}_k^q$ and $\hat{\boldsymbol{s}}_k^q$ introduced in the last expression will be preferred over the general $\boldsymbol{\beta}_k$ notation. This choice more accurately reflects that we are now dealing with sine and cosine Fourier coefficients.

Regarding the weighting functions, the use of complex exponentials (or sine and cosine functions) allows the residual orthogonality conditions in Equation (2.5) to be expressed as follows:

$$\begin{aligned}\hat{\boldsymbol{r}}_k &:= \frac{1}{T}\int_0^T \boldsymbol{r}_\mathrm{h}(t, \{\hat{\boldsymbol{q}}_k\})\,\mathrm{e}^{-ik\Omega t}\,dt = \boldsymbol{0} \quad k = -n_h, \dots, n_h \quad \text{or,} \\ \hat{\boldsymbol{c}}_0^r &:= \frac{1}{T}\int_0^T \boldsymbol{r}_\mathrm{h}(t, \{\hat{\boldsymbol{c}}_k^q, \hat{\boldsymbol{s}}_k^q\})\,dt = \boldsymbol{0} \\ \hat{\boldsymbol{c}}_k^r &:= \frac{1}{T}\int_0^T \boldsymbol{r}_\mathrm{h}(t, \{\hat{\boldsymbol{c}}_k^q, \hat{\boldsymbol{s}}_k^q\})\cos(k\Omega t)\,dt = \boldsymbol{0} \quad k = 1, \dots, n_h \\ \hat{\boldsymbol{s}}_k^r &:= \frac{1}{T}\int_0^T \boldsymbol{r}_\mathrm{h}(t, \{\hat{\boldsymbol{c}}_k^q, \hat{\boldsymbol{s}}_k^q\})\sin(k\Omega t)\,dt = \boldsymbol{0} \quad k = 1, \dots, n_h\end{aligned} \tag{2.7}$$

The left-hand sides of these equations represent the scalar products between the residual $\boldsymbol{r}_\mathrm{h}$ and the Fourier basis functions. By definition, these quantities correspond to the Fourier coefficients of the residuals, denoted as $\hat{\boldsymbol{r}}_k$ or as $\hat{\boldsymbol{c}}_k^r$ and $\hat{\boldsymbol{s}}_k^r$ in the sine-cosine representation. Thus, the weak form requirements expressed in Equation (2.5) reduce, in the context of the harmonic balance method, to the condition that the Fourier coefficients of the residual must be zero. In this case, Equation (2.7) provides a system of $n_q(2n_h+1)$ algebraic equations for the $(2n_h+1)$ unknowns $\hat{\boldsymbol{c}}_k^r$ and $\hat{\boldsymbol{s}}_k^r$.

In Summary — The harmonic balance method is a weighted residual method that transforms a periodic boundary value problem into a system of algebraic equations, which express the requirement that the Fourier coefficients of the residual should be zero. This overarching view of the harmonic balance working principle is illustrated in Figure 2.2.

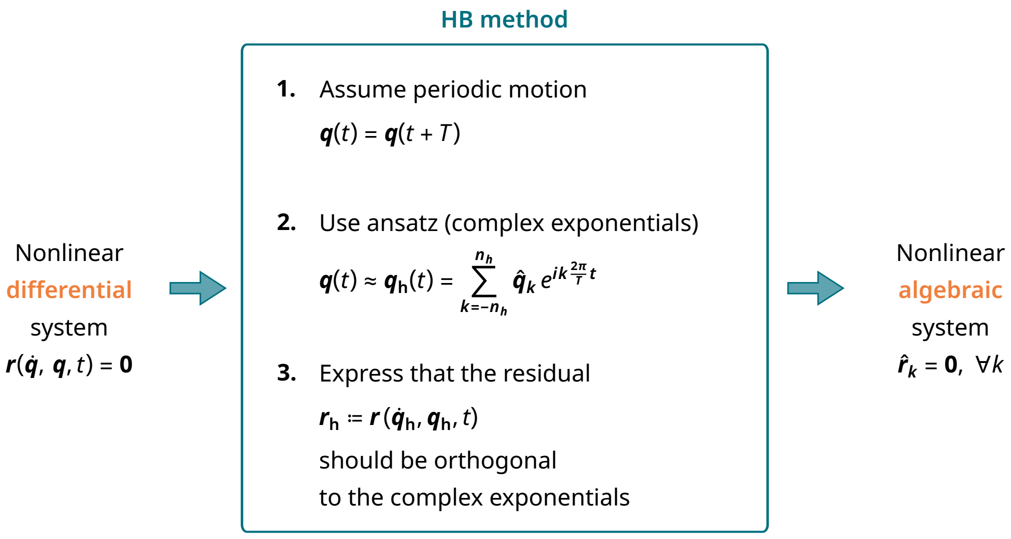

HB method
1. Assume periodic motion
$\boldsymbol{q}(t) = \boldsymbol{q}(t+T)$
2. Use ansatz (complex exponentials)
$\boldsymbol{q}(t) \approx \boldsymbol{q}_{\mathrm{h}}(t) = \sum_{k=-n_h}^{n_h} \hat{\boldsymbol{q}}_k \, e^{ik\frac{2\pi}{T}t}$
3. Express that the residual
$\boldsymbol{r}_{\mathrm{h}} \coloneqq \boldsymbol{r}(\dot{\boldsymbol{q}}_{\mathrm{h}}, \boldsymbol{q}_{\mathrm{h}}, t)$
should be orthogonal
to the complex exponentials
Nonlinear
differential
system
$\boldsymbol{r}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) = \boldsymbol{0}$
Nonlinear
algebraic
system
$\hat{\boldsymbol{r}}_k = \boldsymbol{0}, \ \forall k$

Figure 2.2 Broad view of the harmonic balance working principle.

To simplify the developments that follow, the subsequent short notations will be employed to express the Fourier series in Equation (2.6) and the Fourier coefficients in Equation (2.7):

$$\boldsymbol{q} \approx \boldsymbol{q}_{\mathrm{h}}(\hat{\boldsymbol{q}}_{\mathrm{H}}, t) = \boldsymbol{h}(\Omega t)\,\hat{\boldsymbol{q}}_{\mathrm{H}}\,, \tag{2.8}$$

$$\hat{\boldsymbol{q}}_{\mathrm{H}} = \langle \boldsymbol{h}^*(\Omega t)\,|\,\boldsymbol{q}\rangle\ . \tag{2.9}$$

In Equation (2.8), $\hat{\boldsymbol{q}}_{\mathrm{H}}$ is a column vector of size $n_q(2n_h+1)\times 1$ that contains all the Fourier coefficients of the ansatz. The index H is used to indicate that the underlying quantity aggregates all the harmonics into one vector. When using the sine-cosine representation, the vector $\hat{\boldsymbol{q}}_{\mathrm{H}}$ can be explicitly written as follows:

$$\hat{\boldsymbol{q}}_{\mathrm{H}} = \left[(\hat{\boldsymbol{c}}_0^q)^T,\, (\hat{\boldsymbol{c}}_1^q)^T,\, (\hat{\boldsymbol{s}}_1^q)^T,\, \dots\,,\, (\hat{\boldsymbol{c}}_{n_h}^q)^T,\, (\hat{\boldsymbol{s}}_{n_h}^q)^T\right]\,. \tag{2.10}$$

Concerning the matrix $\boldsymbol{h}$, it collects all the Fourier basis functions, specifically the sine and cosine functions, such that its multiplication with $\hat{\boldsymbol{q}}_{\mathrm{H}}$ yields the truncated Fourier series defining the ansatz. This matrix can be further decomposed as follows:

$$\boldsymbol{h} = \check{\boldsymbol{h}} \otimes \mathbb{I}_{n_q}\,, \tag{2.11}$$

where $\mathbb{I}_{n_q}$ is the identity matrix of size $n_q \times n_q$, $\otimes$ denotes the Kronecker product, and $\check{\boldsymbol{h}}$

is a $1 \times (2n_h + 1)$ row vector containing the sine and cosine basis functions:

$$\check{\boldsymbol{h}} = [1, \cos(\Omega t), \sin(\Omega t), \ldots, \cos(n_h \Omega t), \sin(n_h \Omega t)] \,. \tag{2.12}$$

Thus, the matrix $\boldsymbol{h}$ is of size $n_q \times n_q(2n_h + 1)$.

In Equation (2.9), the notation $\langle a \,|\, b \rangle$ denotes the scalar product of the two vectors a and b, while the quantity $\boldsymbol{h}^*$ represents the Hermitian transpose of $\boldsymbol{h}$. Thus, Equation (2.9) expresses the scalar product between the solution $\boldsymbol{q}$ and the Fourier basis functions. In other words, this product defines the Fourier coefficients $\hat{\boldsymbol{q}}_{\mathrm{H}}$ of the solution.

With these newly introduced short notations, it is possible to succinctly express the orthogonality conditions from Equation (2.7). These conditions can be written as:

$$\hat{\boldsymbol{r}}_{\mathrm{H}}(\hat{\boldsymbol{q}}_{\mathrm{H}}) = \boldsymbol{0} \,. \tag{2.13}$$

To further develop this equation, it is important to note that the differentiation of the ansatz with respect to the independent variable q can be easily obtained. Since $\boldsymbol{q}_{\mathrm{h}}$ is a truncated Fourier series, which is a linear combination of sine and cosine functions, it follows that:

$$\begin{aligned} \dot{\boldsymbol{q}} \approx \dot{\boldsymbol{h}}(\Omega t)\, \hat{\boldsymbol{q}}_{\mathrm{H}} &= \Omega\, \nabla\, \boldsymbol{h}(\Omega t)\, \hat{\boldsymbol{q}}_{\mathrm{H}} \\ &= \left[\Omega\, \check{\nabla}\, \check{\boldsymbol{h}}(\Omega t) \otimes \mathbb{I}_{n_q}\right] \hat{\boldsymbol{q}}_{\mathrm{H}} \,. \end{aligned} \tag{2.14}$$

The operators ∇ and $\check{\nabla}$ used in this equation are functions solely of n_h, and they can be constructed as follows:

$$\begin{gathered} \nabla = \check{\nabla} \otimes \mathbb{I}_{n_q} \,, \quad \text{where} \\ \check{\nabla} = \begin{bmatrix} 0 & & & & \\ & \ddots & & & \\ & & \check{\nabla}_k & & \\ & & & \ddots & \\ & & & & \check{\nabla}_{n_h} \end{bmatrix} \quad \text{and} \quad \check{\nabla}_k = \begin{bmatrix} 0 & -k \\ k & 0 \end{bmatrix} . \end{gathered} \tag{2.15}$$

It thus appears that the use of Fourier series in the harmonic balance method results in a straightforward expression for the differential operator. This operator can indeed be implemented using a sparse matrix of integers.

Finally, Equation (2.13) can be developed as follows:

$$\begin{aligned} \hat{\boldsymbol{r}}_{\mathrm{H}}(\hat{\boldsymbol{q}}_{\mathrm{H}}) &= \langle \boldsymbol{h}^* \,|\, \boldsymbol{r}(\dot{\boldsymbol{q}}_{\mathrm{h}}, \boldsymbol{q}_{\mathrm{h}}, t) \rangle \\ &= \langle \boldsymbol{h}^* \,|\, \boldsymbol{r}(\boldsymbol{h}\, \Omega \nabla \hat{\boldsymbol{q}}_{\mathrm{H}}, \boldsymbol{h}\, \hat{\boldsymbol{q}}_{\mathrm{H}}, t) \rangle \\ &= \frac{1}{T} \int_0^T \boldsymbol{h}^*\, \boldsymbol{r}(\boldsymbol{h}\, \Omega \nabla \hat{\boldsymbol{q}}_{\mathrm{H}}, \boldsymbol{h}\, \hat{\boldsymbol{q}}_{\mathrm{H}}, t)\, dt \\ &= \boldsymbol{0} \,. \end{aligned} \tag{2.16}$$

The requirement to satisfy these conditions on the residual, as expressed in this equation, is referred to as *classical harmonic balance*. Similar to Equation (2.13), this still represents a weak form of the original problem, which manifests as a nonlinear system of $n_q(2n_h+1)$

algebraic equations that must be solved for the unknown Fourier coefficients of the ansatz $\hat{\boldsymbol{q}}_{\mathrm{H}}$.

As mentioned in Section 2.1.1, it is important to remember that a harmonic balance software computes these Fourier coefficients solely to ensure that the weak form is satisfied. The computed ansatz remains an approximation of the solution to the original problem. To enhance the accuracy of the computed solution, one can try to increase the number of terms in the truncated Fourier series of the ansatz. The rate of convergence of the computed ansatz towards the solution $\boldsymbol{q}$ as the number of harmonics n_h increases is discussed in, for example, [9]. Briefly stated, this convergence rate depends on the degree of smoothness of the solution. These convergence aspects will be further explored in Chapter 3.

Other Weighted Residual Methods

As explained in Section 2.1.1, weighted residual methods utilize a predefined set of basis functions $b_k(t)$ to represent the ansatz $\boldsymbol{q}_{\mathrm{h}}$, along with a selected set of weight functions $\rho_j(t)$ for formulating the weak form requirements.

As noted in the previous section, the harmonic balance method employs complex exponentials, or sine and cosine functions, for both the basis and weight functions. For the sake of classification, one can to mention that when a weighted residual method uses the same functions for both the basis and the weight functions, it is referred to as a Galerkin method. Thus, the harmonic balance method is an instance of a Galerkin method.

These two choices of basis and weight functions constitute free parameters that allow for the creation of various weighted residual-based methods, leading to a theoretically infinite number of combinations. The choice of functions made for the harmonic balance method is not the only interesting option; other combinations of basis and weight functions have been explored.

For example, the trigonometric collocation method also uses complex exponentials for the basis functions but employs Dirac delta distributions as weight functions. Referring to the weak form of a weighted residual method expressed in Equation (2.5), it can be deduced that this approach imposes that the residual $\boldsymbol{r}_{\mathrm{h}}$ must be zero only at specific points t_j, known as collocation points. Any weighted residual method that formulates its weak form conditions using collocation points is termed a collocation method. The operational principle of a collocation method is illustrated in Figure 2.3.

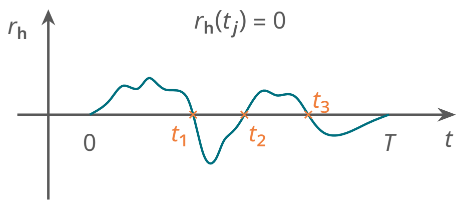


Figure 2.3 Illustration of the working principle of a collocation method. Drawing reproduced and adapted from Krack and Gross [9].

Another popular type of weighted residual method is orthogonal collocation, which uses Lagrange polynomials as basis functions. This method is employed in several software applications, one example being MatCont [29]. Orthogonal collocation is particularly well-suited for stiff problems; however, it does not scale well for large systems [6].

2.1.3 Application to Mechanical Systems

The general formulation of the harmonic balance method has been derived in Equation (2.16) in the preceding section. As mentioned in Chapter 1, the harmonic balance method has found applications across various fields, provided that the problem can be expressed as a periodic boundary problem. This thesis, however, focuses on structural dynamics, particularly in the context of the initial rotordynamic system under study. Moreover, the software utilized for the analysis, pyHarm, is oriented towards structural dynamics. Consequently, this section presents the mathematical derivation of the classical harmonic balance formulation specifically for mechanical systems.

The periodicity constraints outlined in Equation (2.2) remain unchanged. In this context, the independent variable t represents time. The generic form of the differential equation stated in Equation (2.1) will take the specific form:

$$\boldsymbol{M}\ddot{\boldsymbol{q}} + \boldsymbol{C}\dot{\boldsymbol{q}} + \boldsymbol{K}\boldsymbol{q} + \boldsymbol{f}_{\mathrm{nl}}(\ddot{\boldsymbol{q}}, \dot{\boldsymbol{q}}, \boldsymbol{q}) = \boldsymbol{f}_{\mathrm{ext}}(t)\,. \tag{2.17}$$

This mathematical representation of a mechanical system models the dynamic equilibrium of forces acting within the system. In this representation, $\boldsymbol{q}$ denotes the generalized coordinates of the system, while $\boldsymbol{M}$, $\boldsymbol{C}$, and $\boldsymbol{K}$ represent the structural mass, damping, and stiffness matrices, respectively. The first three terms of this equation constitute the linear part of the model. For an explanation of the derivation of these matrices, the reader may refer to Geradin and Rixen [1].

The nonlinearities of the system are expressed by the generic term $\boldsymbol{f}_{\mathrm{nl}}$, which may represent any nonlinear function of time, the generalized coordinates, and their derivatives. Finally, the system may be subjected to an external force, denoted as $\boldsymbol{f}_{\mathrm{ext}}(t)$. This excitation force is independent of the generalized coordinates, and there are no *a priori* restrictions on its expression. The only requirement is that the periodicity condition on the solution must ultimately be satisfied, as stated in Equation (2.2).

Noting f_{lin} as the linear part of the model, Equation (2.17) can be written in a form closer to Equation (2.1):

$$\boldsymbol{r}(\ddot{\boldsymbol{q}}, \dot{\boldsymbol{q}}, \boldsymbol{q}, t) = f_{\mathrm{lin}}(\ddot{\boldsymbol{q}}, \dot{\boldsymbol{q}}, \boldsymbol{q}) + f_{\mathrm{nl}}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) - f_{\mathrm{ext}}(t)\,. \tag{2.18}$$

Injecting an ansatz $\boldsymbol{q}_{\mathrm{h}}(\hat{\boldsymbol{q}}_{\mathrm{H}}, t)$ into this equation leads to rewriting the weak form of the HB method as follows:

$$\begin{gathered}\hat{\boldsymbol{f}}_{\mathrm{lin,H}}(\hat{\boldsymbol{q}}_{\mathrm{H}}, \Omega) + \hat{\boldsymbol{f}}_{\mathrm{nl,H}}(\hat{\boldsymbol{q}}_{\mathrm{H}}, \Omega) - \hat{\boldsymbol{f}}_{\mathrm{ext,H}} = \mathbf{0}\,,\ \text{or} \\ \langle \boldsymbol{h}^* \,|\, \boldsymbol{r}(\ddot{\boldsymbol{q}}_{\mathrm{h}}, \dot{\boldsymbol{q}}_{\mathrm{h}}, \boldsymbol{q}_{\mathrm{h}}, t)\rangle = \mathbf{0}\,.\end{gathered} \tag{2.19}$$

Recalling the differentiation rules established in Equation (2.14), the last equation can be further developped into:

$$\left\langle \boldsymbol{h}^* \,|\, \boldsymbol{r}(\Omega^2\nabla^2\boldsymbol{h}\hat{\boldsymbol{q}}_{\mathrm{H}}, \Omega\nabla\boldsymbol{h}\hat{\boldsymbol{q}}_{\mathrm{H}}, \hat{\boldsymbol{q}}_{\mathrm{H}}, t)\right\rangle = \mathbf{0}\,. \tag{2.20}$$

The equations can be further developed by detailing the expressions of the three terms that constitute the linear part of the model. It can be demonstrated, without proof, that:

$$\begin{aligned}\langle \boldsymbol{h}^* \,|\, \boldsymbol{M}\ddot{\boldsymbol{q}}_{\mathrm{h}}\rangle &= \left(\check{\nabla}^2 \otimes \Omega^2 \boldsymbol{M}\right) \hat{\boldsymbol{q}}_{\mathrm{H}} \\ \langle \boldsymbol{h}^* \,|\, \boldsymbol{C}\dot{\boldsymbol{q}}_{\mathrm{h}}\rangle &= \left(\check{\nabla} \otimes \Omega \boldsymbol{C}\right) \hat{\boldsymbol{q}}_{\mathrm{H}} \\ \langle \boldsymbol{h}^* \,|\, \boldsymbol{K}\boldsymbol{q}_{\mathrm{h}}\rangle &= \left(\mathbb{I}_{2n_h+1} \otimes \boldsymbol{K}\right) \hat{\boldsymbol{q}}_{\mathrm{H}}\end{aligned} \tag{2.21}$$

Consequently, the weak form of Equation (2.19) can be expressed as:

$$\left(\check{\nabla}^2 \otimes \Omega^2 \boldsymbol{M} + \check{\nabla} \otimes \Omega \boldsymbol{C} + \mathbb{I}_{2n_h+1} \otimes \boldsymbol{K}\right) \hat{\boldsymbol{q}}_{\mathrm{h}} + \hat{\boldsymbol{f}}_{\mathrm{nl}}(\hat{\boldsymbol{q}}_{\mathrm{h}}, \Omega) - \hat{\boldsymbol{f}}_{\mathrm{ext}}(\Omega) = \boldsymbol{0}\,, \tag{2.22}$$

or, in a more concise form:

$$\hat{\boldsymbol{r}}(\hat{\boldsymbol{q}}, \Omega) = \boldsymbol{S}(\Omega)\hat{\boldsymbol{q}} + \hat{\boldsymbol{f}}_{\mathrm{nl}}(\hat{\boldsymbol{q}}, \Omega) - \hat{\boldsymbol{f}}_{\mathrm{ext}}(\Omega) = \boldsymbol{0}\,. \tag{2.23}$$

This last equation will be referred to as the *HB equations* in the following. For clarity, the H indices have been omitted in this last equation. In subsequent discussions, only aggregated quantities will be considered, and the H will not always be explicitly stated.

This final equation represents the reformulation of the classical HB weak form within the context of a mechanical system described by Equation (2.17). It defines the system of nonlinear equations that a structural HB solver will ultimately address. These equations thus have to be solved iteratively with, *e.g.*, a Newton-Raphson procedure. The resolution of this set of nonlinear algebraic equations will be the focus of Section 2.2.

2.1.4 Alternating Frequency-Time Scheme

In the weak form of the structural problem established in Equation (2.22), it is evident that the nonlinear force term depends on the sought Fourier coefficients. Consequently, when iteratively solving the weak form equations, this nonlinear term must be evaluated at each iteration using the current values of $\hat{\boldsymbol{q}}_{\mathrm{H}}$.

In certain straightforward cases, it is possible to derive an analytical expression for $\hat{\boldsymbol{f}}_{\mathrm{nl}}$. For instance, polynomial forces can be expanded into a Fourier series [9].

However, for a generic nonlinear force, obtaining an analytical form of $\hat{\boldsymbol{f}}_{\mathrm{nl}}$ is not feasible. In such cases, the most commonly employed technique for evaluating the nonlinear force is the alternating frequency-time (AFT) domain method. This technique was first introduced by Cameron and Griffin [30] and involves evaluating the nonlinear forces based on the ansatz in the time domain.

The expression for the nonlinear force $\boldsymbol{f}_{\mathrm{nl}}$ in the time domain, as described in Equation (2.17), is known. To evaluate $\hat{\boldsymbol{f}}_{\mathrm{nl}}$, one can first convert the coefficients $\hat{\boldsymbol{q}}_{\mathrm{H}}$ into the time domain using an inverse fast Fourier transform (FFT). The nonlinear forces can then be computed in the time domain. Subsequently, this nonlinear force can be transformed back into the frequency domain using an FFT. This process is illustrated in Figure 2.4.

It is important to note that the FFT operation can be applied quite efficiently. To understand this, one must recognize that time-domain signals cannot be perfectly represented numerically. The harmonic balance software that processes these time signals utilizes a finite number n_t of time values over the period T to sample the signals. The sampled approximation of a time signal $\boldsymbol{f}(t)$ is denoted as $\tilde{\boldsymbol{f}}(t)$.

It can be shown [9, 6] that the FFT and its inverse can be expressed as linear operators, allowing for the transition between time-domain and frequency-domain representations

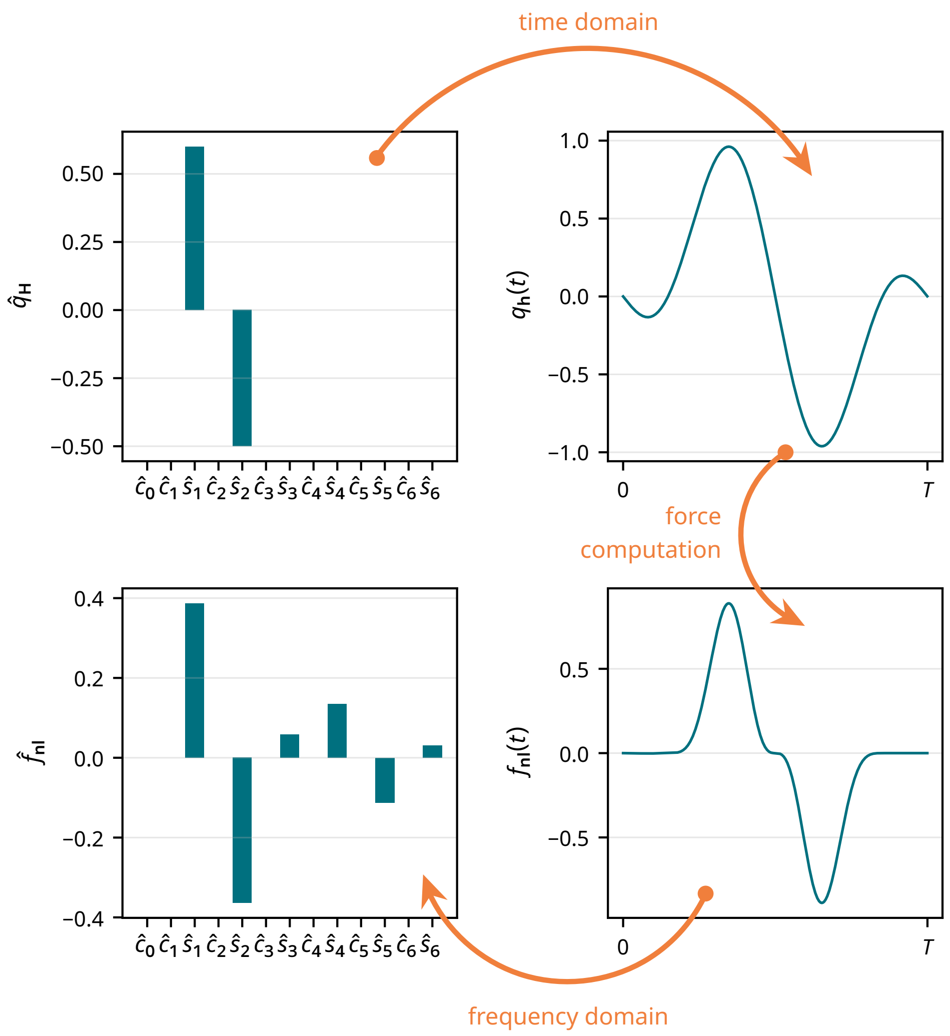


Figure 2.4 Illustration of the working principle of the alternating frequency-time procedure. This figure was generated using the FFT operators provided by pyHarm (function `DynamicOperator.compute_DFT()`, discussed in Appendix B). The nonlinear force depicted here is expressed as $\boldsymbol{f}_\mathrm{nl}(t) = \boldsymbol{q}^3(t)$, which is similar to the behavior of a Duffing oscillator.

of a signal as follows:

$$\begin{aligned} \tilde{\boldsymbol{f}}_\mathrm{T} &= \boldsymbol{E}_\mathrm{TH}\,\hat{\boldsymbol{f}}_\mathrm{H} \\ \hat{\boldsymbol{f}}_\mathrm{H} &= \boldsymbol{E}^*_\mathrm{HT}\,\tilde{\boldsymbol{f}}_\mathrm{T}\,. \end{aligned} \tag{2.24}$$

Similar to the index H, the index T indicates that the values of $\tilde{\boldsymbol{f}}$ for each time sample are stacked into a single column vector of size $n_q n_t \times 1$. The operators $\boldsymbol{E}_\mathrm{TH}$ and $\boldsymbol{E}^*_\mathrm{HT}$ are matrices of sizes $n_q n_t \times (2n_h + 1)$ and $(2n_h + 1) \times n_q n_t$, respectively, and they actually consist of sine and cosine function values evaluated at the time samples. Consequently, these matrices are fully defined by the dimensions n_q, n_h and n_t, making their numerical implementation relatively straightforward.

These operators can then be utilized to rewrite the AFT procedure as follows:

$$\hat{\boldsymbol{f}}_\mathrm{nl,H}(\hat{\boldsymbol{q}}_\mathrm{H},\,\Omega) \approx \hat{\boldsymbol{f}}^\mathrm{AFT}_\mathrm{nl,H}(\hat{\boldsymbol{q}}_\mathrm{H},\,\Omega) = \boldsymbol{E}^*_\mathrm{HT}\,\tilde{\boldsymbol{f}}_\mathrm{nl,T}(\boldsymbol{E}_\mathrm{TH}\,\nabla\,\Omega\,\hat{\boldsymbol{q}}_\mathrm{H},\,\boldsymbol{E}_\mathrm{TH}\,\hat{\boldsymbol{q}}_\mathrm{H})\,. \tag{2.25}$$

In summary, the AFT procedure for evaluating the nonlinear forces can be accomplished through simple matrix multiplications. In fact, it has been shown that the AFT method can be even more efficient than using an analytical expression for $\hat{\boldsymbol{f}}_{\text{nl}}$ [31].

2.2 Solving The HB Equations

The harmonic balance method results in a weak form of the original problem, which manifests as a system of algebraic equations to be solved, referred to as the HB equations. Of course, the HB method can be effectively applied to linear systems. In such cases, the HB equations take the form of a linear algebraic system, reverting to the conventional linear frequency domain analysis commonly used in mechanics and electrical circuit analysis for computing the steady-state response of these systems.

However, the HB method reveals its utility during the analysis of nonlinear systems. In this scenario, the HB equations are nonlinear and must be solved iteratively. Procedures such as Newton-Raphson or hybrid Powell can be employed.

This section will first explain how to compute a solution of the HB equations using the Newton-Raphson procedure for a specific excitation frequency. Subsequently, it will demonstrate how to solve the HB equations across a range of excitation frequencies, thereby computing the frequency response of the system.

2.2.1 Computing One Solution Point

When solving the HB equations at a specific frequency Ω, the Fourier coefficients of the residual, $\hat{\boldsymbol{r}}$, are solely functions of the sought solution, $\hat{\boldsymbol{q}}$. Thus, $\hat{\boldsymbol{r}}$ represents a mapping from $\mathbb{R}^{n_q(2n_h+1)}$ to $\mathbb{R}^{n_h(2n_h+1)}$.

The Newton-Raphson procedure is an iterative method suitable for solving this type of system. The approach begins with an initial guess $\hat{\boldsymbol{q}}^{(0)}$ for the solution. The solution $\hat{\boldsymbol{q}}^{(i+1)}$ at the next iteration is predicted from the current solution $\hat{\boldsymbol{q}}^{(i)}$ using the following equation:

$$\hat{\boldsymbol{q}}^{(i+1)} = \hat{\boldsymbol{q}}^{(i)} - \partial_{\hat{\boldsymbol{q}}}\hat{\boldsymbol{r}}(\hat{\boldsymbol{q}}^{(i)})^{-1}\,\hat{\boldsymbol{r}}(\hat{\boldsymbol{q}}^{(i)})\,, \tag{2.26}$$

where $\partial_{\hat{\boldsymbol{q}}}\hat{\boldsymbol{r}}$ denotes the Jacobian matrix of $\hat{\boldsymbol{r}}$. To make the next prediction, this method thus merely consists in linearizing the problem around the previously predicted solution through a Tayor expansion of the residual.

The choice of the initial guess $\hat{\boldsymbol{q}}^{(0)}$ is crucial. For the Newton-Raphson method to converge properly towards a solution, the initial guess should be selected carefully, ensuring it already is sufficiently close to the sought solution. When this condition is met, the method is known for its rapid convergence, and it has been widely applied in various numerical software that addresses systems of nonlinear equations.

It is also important to note that estimating the Jacobian $\partial_{\hat{\boldsymbol{q}}}\hat{\boldsymbol{r}}$ at each iteration is not computationally trivial. In an HB solver like pyHarm, a significant portion of the computation time is dedicated to these Jacobian evaluations. As discussed in more detail in Section A.1, a substantial part of the pyHarm code is focused on constructing the Jacobian of the entire mechanical system under study and evaluating it as efficiently as possible.

The most straightforward method for evaluating the Jacobian is through finite differences. In this case, an approximation of the Jacobian is obtained at a point $\hat{\boldsymbol{q}}$ by evaluating the residual $\hat{\boldsymbol{r}}$ in the neighborhood of that point.

Finite differences have the advantage of being easy to implement; however, since evaluating the Jacobian is a critical step in the HB solver, one might consider a second option known as *automatic differentiation*. This more sophisticated technique aims to evaluate the derivative of a function accurately and efficiently by applying the chain rule of calculus on the function to differentiate, down to a very granular level. In particular, pyHarm utilizes the Python Jax library, which provides automatic differentiation utilities. When feasible, automatic differentiation is employed to compute the Jacobians.

2.2.2 Numerical Continuation

In many research fields involving the study of dynamical systems, it is critical to assess the frequency response of the system. This is particularly relevant to the initial problem of this thesis: it is essential to ensure that the vibrations of a rotordynamic system remain within tolerable limits across a certain range of excitation frequencies.

This leads to the desire to solve the HB equations expressed in Equation (2.22) for an entire range of excitation frequencies. More concretely, the problem under consideration can be formalized as follows [9]:

$$\begin{aligned} \textsf{solve} \quad & \hat{\boldsymbol{r}}(X) = 0\,, \\ \textsf{with respect to} \quad & X = [\hat{\boldsymbol{q}}^T,\, \mathit{\Omega}]^T\,, \\ \textsf{where} \quad & \hat{\boldsymbol{r}}, \hat{\boldsymbol{q}} \in \mathbb{R}^{n_q(2n_h+1)\times 1}\,, \\ \textsf{in the interval} \quad & \mathit{\Omega}_\mathrm{s} \leq \mathit{\Omega} \leq \mathit{\Omega}_\mathrm{e}\,. \end{aligned} \tag{2.27}$$

A straightforward initial approach would involve defining a sample of ordered frequencies within the range $\mathit{\Omega}_\mathrm{s}$ to $\mathit{\Omega}_\mathrm{e}$ and solving the HB equations for each solution point. The solution computed at a specific frequency can then serve as the initial guess for the next frequency in the sample. This procedure is referred to as *sequential continuation*.

In linear mechanics, the frequency response of the system is characterized by a one-to-one function that maps the excitation frequency to the amplitude level experienced by a selected DOF of the system. However, as qualitatively discussed in Section 1.1.2, the frequency response of a nonlinear system can exhibit much more complex behaviors. It was shown in Figure 1.1 that the HB equations can possess multiple solutions for a single frequency. Consequently, the system's response is no longer a one-to-one function with respect to frequency. The solution to the problem presented in Equation (2.27) thus graphically takes the form of a curve in the frequency-amplitude space, referred to as a nonlinear frequency response curve (NFRC).

One consequence of this complexity is that, to solve the problem in Equation (2.27), one cannot simply define a sample of frequencies between $\mathit{\Omega}_\mathrm{s}$ and $\mathit{\Omega}_\mathrm{e}$ and solve the HB equations independently for each frequency. Referring again to the Duffing example in Figure 1.1, it was shown that the state to which the system eventually stabilizes may depend on the initial conditions. Therefore, solving the HB equations only once for each sampled frequency will yield only one solution, which will depend on the initial guess provided to the iterative solver.

Thus, more robust techniques than sequential continuation are required, allowing for the tracking of a branch of solutions. This area of study is known as numerical continuation.

Homotopy

Many of the techniques presented in Chapter 4 rely on the numerical continuation algorithms discussed in Allgower and Georg [4]. In this book, numerical continuation is introduced in a context that is not directly related to tracking a NFRC. Specifically, the problem addressed in this book involves solving a generic nonlinear system:

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0}\,, \tag{2.28}$$

where the sought solution is denoted $\bar{\boldsymbol{x}}$ and $\boldsymbol{F}$ is a mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$. As mentioned in the previous section, such a system can be solved using a Newton-Raphson procedure. However, it was also noted that this iterative method may struggle to converge if a suitable initial guess cannot be found. To address this issue, a proposed solution is to use a *homotopy*:

$$\boldsymbol{H}(\boldsymbol{x}, \lambda) := (1-\lambda)\,\boldsymbol{F}(\boldsymbol{x}_0) + \lambda\,\boldsymbol{G}(\boldsymbol{x})\,, \tag{2.29}$$

In this expression, $\boldsymbol{G}$ is a selected function for which it is known that $\boldsymbol{x}_0$ constitutes a zero point. The function $\boldsymbol{H}(\boldsymbol{x}, \lambda)$ defines an affine combination of $\boldsymbol{F}$ and $\boldsymbol{G}$, with λ referred to as the homotopy parameter. The idea is to start at the point $(\boldsymbol{x}, \lambda) = (\boldsymbol{x}_0, 1)$ and iteratively compute the solution of $\boldsymbol{H}$. This process is repeated for values of the homotopy parameter that gradually decrease to zero, ultimately leading to a solution of Equation (2.28).

In other words, this technique involves tracing an implicitly defined curve $\boldsymbol{c}(s) \in \boldsymbol{H}^{-1}(\boldsymbol{0})$, starting from the point $(\boldsymbol{x}_0, 1)$ and moving towards a solution point $(\bar{\boldsymbol{x}}, 0)$. This process is illustrated in Figure 2.5.

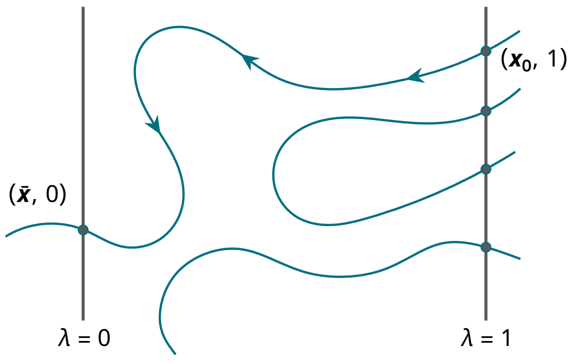


Figure 2.5 Illustration of the solution-finding process *via* a continuous homotopy deformation. Reproduced and adapted from Allgower and Georg [4].

Taking a step back, it becomes clear that the homotopy deformation technique presented in this book is analogous to the problem of interest here. More specifically, the desire to follow the curve $\boldsymbol{c}(s) \in \boldsymbol{H}^{-1}(\boldsymbol{0})$ is similar to the goal of tracking the NFRC of the system, which is implicitly defined by $\hat{\boldsymbol{r}}^{-1}(\boldsymbol{0})$. In fact, the function $\boldsymbol{H}(\boldsymbol{x}, \lambda)$ is analogous to $\hat{\boldsymbol{r}}(\hat{\boldsymbol{q}}, \Omega)$, where Ω can be viewed as the homotopy parameter that varies over a specified range.

Thus, all the numerical continuation techniques presented in Allgower and Georg [4] are almost directly applicable to solving the actual problem outlined in Equation (2.27).

Predictor-Corrector Method

Basic sequential continuation has been shown to be insufficient for effectively tracking a NFRC. One alternative continuation technique is the predictor-corrector method, which is the approach implemented in pyHarm. Its working principle is illustrated in Figure 2.6.

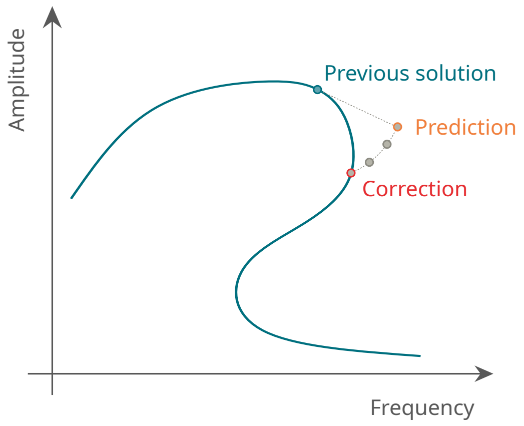


Figure 2.6 Illustration of the working principle of a predictor-corrector continuation on a fictitious nonlinear frequency response curve.

As shown, each successive solution point of the NFRC is determined in two steps. First, a prediction of the next solution is made based on the previously computed solution. Then, a series of corrections are iteratively computed using a procedure such as Newton-Raphson to ensure that the predicted point converges to an actual solution on the curve.

The prediction step typically involves an Euler prediction:

$$X_{\text{pred}}^{(i+1)} = X^{(i)} + h\,\boldsymbol{t}(\partial_X \hat{\boldsymbol{r}}(X^{(i)}))\,, \tag{2.30}$$

where $h > 0$ is a chosen step size, and $\boldsymbol{t}(\partial_X \hat{\boldsymbol{r}}(X^{(i)}))$ is a $n_q(2n_h + 1) + 1 \times 1$ column vector representing the tangent of the NFRC at $X^{(i)}$. This tangent will be defined more precisely in the following discussion.

It is important to note that $\hat{\boldsymbol{r}}(X)$ is a mapping from $\mathbb{R}^{n_q(2n_h+1)+1}$ to $\mathbb{R}^{n_q(2n_h+1)}$. Consequently, the equation $\hat{\boldsymbol{r}}(X) = \boldsymbol{0}$ represents an underdetermined system of nonlinear equations. As a result, the Jacobian $\partial_X \hat{\boldsymbol{r}}$ is a $n_q(2n_h + 1) \times n_q(2n_h + 1) + 1$ matrix, whose kernel typically contains two unit norm vectors, corresponding to the two possible directions for traversing the curve. Therefore, there are two distinct values that can define the tangent $\boldsymbol{t}$.

However, it is essential for the continuation procedure to have a means of following a consistent direction along the curve to ensure that successive predictions are made in the same direction. To achieve this, one can introduce the augmented Jacobian:

$$\begin{pmatrix} \partial_X \hat{\boldsymbol{r}}(\boldsymbol{c}(s)) \\ \dot{\boldsymbol{c}}(s)^* \end{pmatrix} \tag{2.31}$$

where $\boldsymbol{c}(s)$ is a parametrization of the NFRC such that $\boldsymbol{c}(s) \in \hat{\boldsymbol{r}}(\boldsymbol{c}(s)) = \boldsymbol{0}$.

It can be shown [4] that this augmented Jacobian is non-singular along the NFRC, except in special cases. Consequently, its determinant maintains the same sign along the chosen parametrization of the curve. This property allows the determinant to be used to determine the orientation of traversal. This leads to the definition of the tangent vector induced by the Jacobian $\boldsymbol{t}(\partial_X \hat{\boldsymbol{r}})$, or simply the tangent $\boldsymbol{t}$, as follows:

$$\partial_X \hat{\boldsymbol{r}}(\boldsymbol{t}) = \boldsymbol{0}, \quad ||\boldsymbol{t}|| = 1, \quad \det \begin{pmatrix} \partial_X \hat{\boldsymbol{r}} \\ \boldsymbol{t}^* \end{pmatrix} > 0 \,. \tag{2.32}$$

In other words, the NFRC has an orientation, which we arbitrarily define as positive when the determinant of the augmented Jacobian is positive.

It is important to note that finding the tangent of the NFRC involves computing the Jacobian of $\hat{\boldsymbol{r}}$. This emphasizes once more the care that must be taken in calculating these Jacobians.

The Euler prediction made in Equation (2.30) uses the tangent of the NFRC as the direction for finding the next predicted solution. However, this is not the only option. For instance, the secant prediction extrapolates the two most recent solutions to make the next prediction. This method is easier to compute and does not require the evaluation of the Jacobian. Nevertheless, the Jacobians still need to be evaluated for the correction steps. Moreover, the secant prediction generally performs less effectively than the tangent prediction. Both tangent and secant predictors are available in pyHarm.

The underdetermined nature of the system also affects the corrector steps. One way to address this is to add an additional equation, known as a closure equation. The system to solve is now of size $n_q(2n_h + 1) + 1$, where the unknowns include the Fourier coefficients $\hat{\boldsymbol{q}}$ as well as the frequency Ω.

One possible closure equation is the arc length. This imposes a condition to search for a solution that is at the same distance from the last solution as the prediction made. Other possibilities exist as well; for example, pseudo arc length is simpler to implement but less robust. Both the arc length and pseudo arc length methods are implemented in pyHarm.

2.3 Stability and Bifurcation Points

As previously mentioned, the frequency response of a nonlinear system can exhibit much more complex behavior than that of a linear system. It has been demonstrated with a simple Duffing oscillator that the nonlinear frequency response curve (NFRC) is not necessarily a one-to-one function. In fact, this is not the only behavior that distinguishes a NFRC from a linear frequency response. Figure 2.7 qualitatively illustrates some of the notable behaviors that a NFRC can exhibit.

As stated earlier, the frequency response can exhibit turning points along the curve, also known as *fold bifurcations*. Additionally, it is possible for alternative branches of solutions to emerge at certain frequencies. The points where these auxiliary branches emerge are referred to as *branching points*. Solutions to the HB equations may also be found in regions completely detached from the main solution curve, forming what are known as *isolas*.

Fold bifurcations and branching points will be discussed extensively in the following chapters, and a more formal definition is provided hereafter. Regarding potential isolas, they will not be addressed in the subsequent discussions of this thesis. It is worth mentioning that their identification remains challenging. Recently, the use of a machine learning-based technique for isola detection has shown promising results [32].

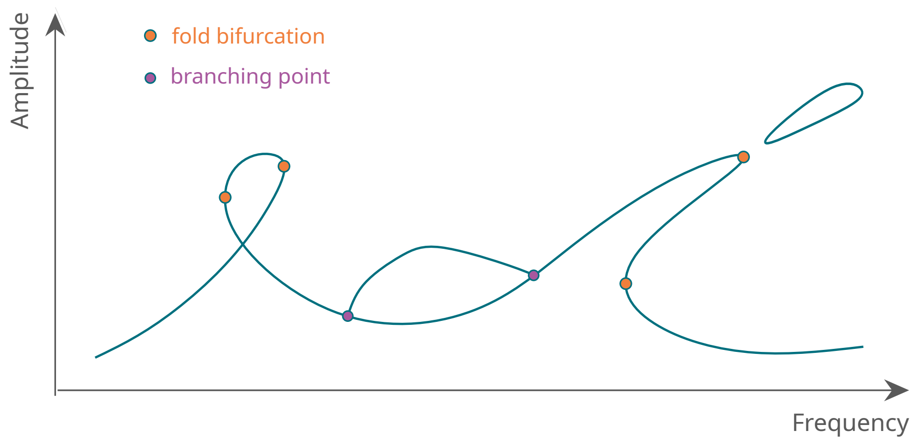


Figure 2.7 Qualitative illustration of some of the noteworthy behaviors that can be exhibited by a nonlinear frequency response curve.

The definitions of fold bifurcation and turning points used in the following sections are based on those established in Detroux et al. [6].

More specifically, a fold bifurcation is said to occur when the Jacobian $\partial_{\hat{q}}\hat{\boldsymbol{r}}$ changes sign. A fold bifurcation can thus be detected when:

$$\det(\partial_{\hat{q}}\hat{\boldsymbol{r}}) = 0\,. \tag{2.33}$$

Additionally, a branching point is said to occur when the augmented Jacobian changes sign, which can be detected using:

$$\det\begin{pmatrix}\partial_X\hat{\boldsymbol{r}}\\ \boldsymbol{t}^*\end{pmatrix} = 0\,. \tag{2.34}$$

Other types of bifurcations, such as second-order Hopf (or Neimark-Sacker) bifurcations, will not be covered here.

It is important to note that the bifurcation definitions and classifications provided in Detroux et al. [6] result from a stability analysis of the Floquet exponents of the system. These exponents can be obtained *via* Hill's method [11, 6, 13, 21], which was first introduced in Von Groll and Ewins [18]. Specifically, a *bifurcation* occurs whenever the system changes its stability regime [9, 6].

However, stability analysis will not be addressed in this thesis. Readers seeking an in-depth analysis of system stability and bifurcations can refer to Seydel [17] and Kuznetsov [10].

3
Initial Problem Analysis

Now that the theory of harmonic balance and continuation methods has been reviewed, it is possible to describe and discuss the problem that serves as the foundation for this thesis: the analysis of the forced response of a rotordynamic model provided by Safran Tech.

This section begins by specifying the mathematical formulation of the rotordynamic system and explains how it can be solved through an HB numerical continuation.

The following section will present the response of the rotor system computed by pyHarm. It will be demonstrated that the numerical continuation fails when an asymmetric bearing clearance is applied to the model.

Finally, a short-term solution is proposed in the third section. This solution allows for the successful completion of the continuation for the problematic case. However, it is important to note that this solution addresses only this specific issue. It will then be concluded that a more robust, long-term solution needs to be implemented in pyHarm.

3.1 Mathematical Formulation

As previously explained in Chapter 1, Safran Tech encountered issues when simulating rotordynamic systems with their experimental HB solver, pyHarm. More specifically, this solver is unable to perform numerical continuation of such systems properly under certain circumstances.

To investigate this issue, Safran Tech provided a simplified rotor model, which has been described in Section 1.2.1. As illustrated in the graph, the rotor toy model consists of a discretized structure with six nodes. Each of these six nodes has six degrees of freedom, allowing for translation and rotation in three-dimensional (3D) space. Consequently, the toy model can be represented as a 36 degrees-of-freedom system, modeled as follows [3]:

$$\boldsymbol{M}\,\ddot{\boldsymbol{q}} + \Omega\,\boldsymbol{G}\,\dot{\boldsymbol{q}} + \boldsymbol{C}\,\dot{\boldsymbol{q}} + \boldsymbol{K}\,\boldsymbol{q} + f_{\mathrm{nl}}(\boldsymbol{q},\, t) = \Omega^2\,\boldsymbol{\phi}_{\mathrm{unb}}(t)\,. \tag{3.1}$$

In this equation, Ω denotes the rotation speed of the shaft. The terms $\boldsymbol{M}$, $\boldsymbol{G}$, $\boldsymbol{C}$, and $\boldsymbol{K}$ represent the global mass, gyroscopic, damping, and stiffness matrices of the system, respectively. The first four terms of this equation constitute the linear part of the model. The nonlinear force f_{nl} corresponds to the presence of the clearance gap between the bearing and its housing, and its expression will be detailed below. Finally, the term $\Omega^2\,\boldsymbol{\phi}_{\mathrm{unb}}$ represents the action of the unbalance force on the system. It can be shown [3] that the force induced by an unbalance results in a periodic excitation, with an amplitude proportional to the square of the rotation speed of the shaft.

One can directly observe that this system of equations can be trivially rearranged into the following residue form:

$$\boldsymbol{r}(\dot{\boldsymbol{q}}, \boldsymbol{q}, t) = \boldsymbol{0}\,. \tag{3.2}$$

Assuming that the motion of the rotordynamic system is symmetric, the characteristics of this problem are compatible with the generic form of the boundary value problem outlined in Equations (2.1) and (2.2). This compatibility indicates that the harmonic balance method can indeed be applied to study this rotordynamic system.

In practice, Safran Tech provided the global structural matrices for two cases. The distinguishing factor between these two cases lies in the nonlinear bearing-housing interface located between nodes 5 and 6 of the system. In the first case, the stiffness of this bearing-housing interface is radially symmetric. In the second case, the support is less rigid in the radial direction compared to the direction that is perpendicular to it.

Regarding the nonlinear forces induced by the clearance gap, they can be modeled as follows:

$$\forall t \in [0,\, T], \begin{cases} f_{\mathrm{nl}}(\boldsymbol{q},\, t) = \boldsymbol{0}, & \text{if } \|\Delta\boldsymbol{x}(t)\| - g < 0 \\ f_{\mathrm{nl}}(\boldsymbol{q},\, t) = \gamma \cdot \|\Delta\boldsymbol{x}(t)\| - g, & \text{if } \|\Delta\boldsymbol{x}(t)\| - g >= 0 \end{cases}. \tag{3.3}$$

In this equation, g represents the gap size between the bearing's outer ring and the inner wall of the support. In the present rotor toy model, this gap is set to 0.15 mm. The quantity $\Delta\boldsymbol{x}$ denotes the difference between the radial displacement of the bearing and that of its housing.

This nonlinear force can be interpreted as follows: when the bearing and its housing are not in contact, $\Delta\boldsymbol{x}$ is less than the gap size, and the nonlinear force is set to zero, resulting in linear behavior of the system. However, when contact occurs between the bearing and its support, a large rigidity, denoted as γ in Equation (3.3), is applied between the bearing and the housing, effectively (approximately) preventing the bearing from penetrating into its housing.

In practice, pyHarm offers an object that can implement the nonlinear behavior described in Equation (3.3) (`PenaltyBilateralGap`). Section B.3 demonstrates its usage in the construction of the rotor toy model within pyHarm.

3.2 Frequency Response Analysis

The simulation of the symmetric rotor model using pyHarm is illustrated in Figure 3.1. It is evident that when $|\Delta\boldsymbol{x}|/g$ is below 1, the system exhibits linear behavior. However, a significant change in the system's behavior occurs when contact between the bearing and its support is established.

The computed NFRC indicates that a two fold bifurcations has been detected by pyHarm. The solver's output at these two locations is displayed in Figure 3.2. The continuation method successfully computed the system's response over a frequency range from 1 Hz to 300 Hz.

In addition to the NFRC, the results obtained from pyHarm also allow for the visualization of the rotor's orbital motion. This representation of the system's response is

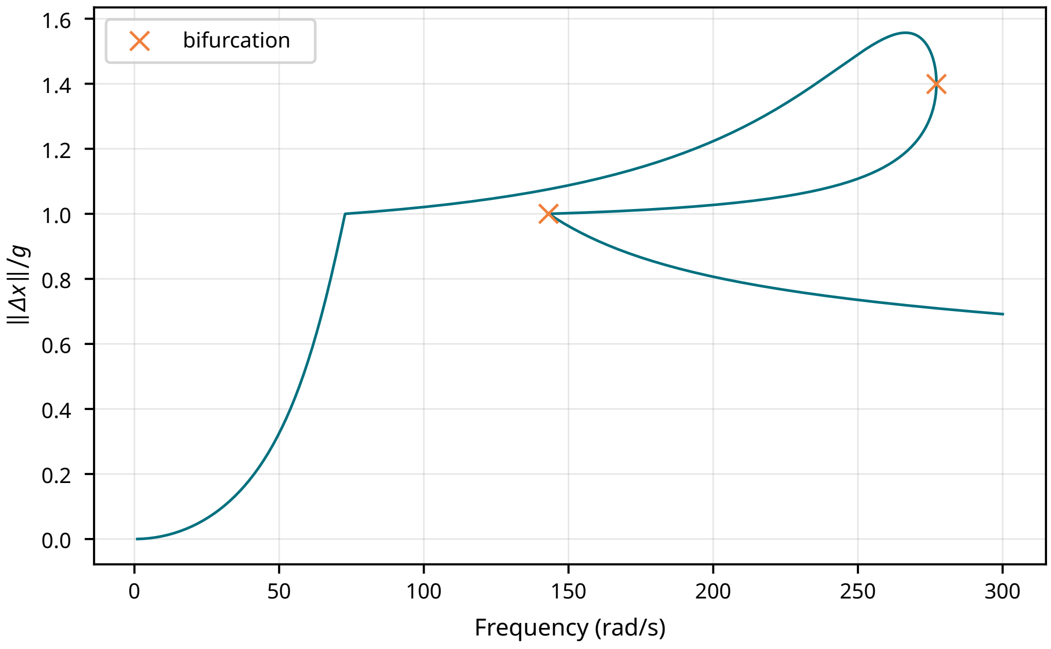


Figure 3.1 Computation of the NFRC of the symmetric rotor model with pyHarm.

```
solution converged at om=277.134927583256
solution converged at om=277.1357168027481
Warning: a limit point (fold bifurcation) was detected ⟶ path direction is reversed
solution converged at om=277.12824671483367
solution converged at om=277.11237748573825
...
solution converged at om=143.18967881881346
solution converged at om=143.09141053186187
Warning: a limit point (fold bifurcation) was detected ⟶ path direction is reversed
solution converged at om=143.19138800607448
solution converged at om=143.2913446879394
```

Figure 3.2 Console log indicating the reversal of path direction at the two bifurcation points for the symmetric model.

shown in Figure 3.3. Detailed instructions on generating these orbit plots are provided in Section B.1.2.

The observed motion of the shaft du to the unbalance force, at the location of the bearing-housing interface, consists in regular circular rotation.

Figure 3.4 presents the simulation of the asymmetric rotor model. It is observed that the continuation procedure fails at a certain point.

Upon examining the close view of the NFRC in Figure 3.4, it becomes evident that a third bifurcation point has been detected, at which the continuation fails.

Similar to the symmetric model, it is possible to visualize the orbital motion of the rotor at the location of the nonlinear bearing-housing support for the asymmetric case. The results are displayed in Figure 3.5.

In contrast to the symmetric model, the motion of the shaft at the configuration of the detected bifurcation is no longer characterized by regular circular motion. Instead, the motion is more complex and exhibits partial rubbing between the bearing and the housing.

The observations from Figures 3.4 and 3.5 suggest that the asymmetric rotordynamic

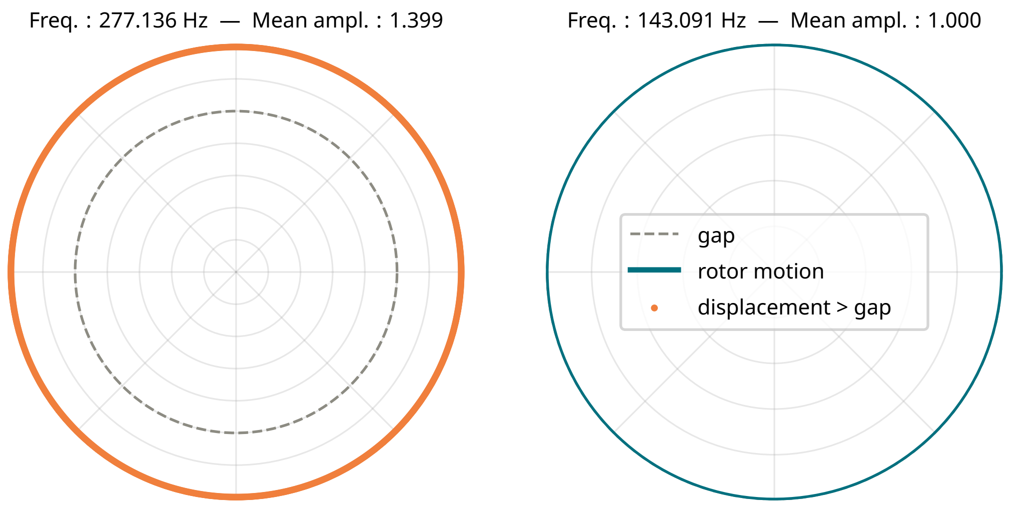


Figure 3.3 Plots of the rotor orbital motion, at the location of the two detected bifurcation points.

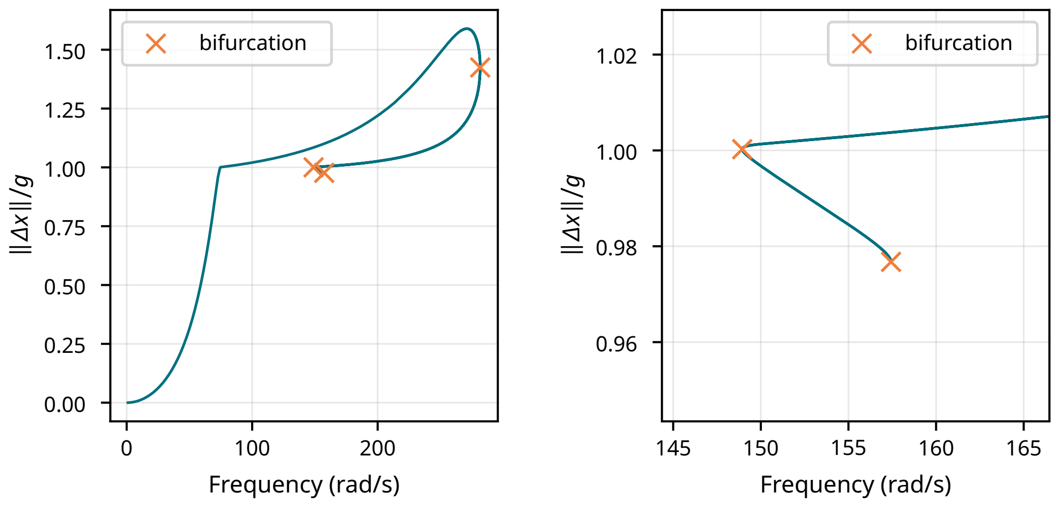


Figure 3.4 Computation of the NFRC of the asymmetric rotor model with pyHarm. The right graph is simply a closer view of the left one.

system experiences a branching of the main solution curve at this location. It appears that pyHarm encounters significant difficulties in managing this bifurcation point.

The challenges faced by the numerical continuation are illustrated in Figure 3.6.

The prediction steps computed by the continuation are also displayed. It is evident that these predictions exhibit erratic behavior. Furthermore, in this failing region, the nonlinear solver of pyHarm experiences significant convergence difficulties, as shown in Figure 3.7.

All preceding observations suggest that the bifurcation handling capabilities of pyHarm could be improved.

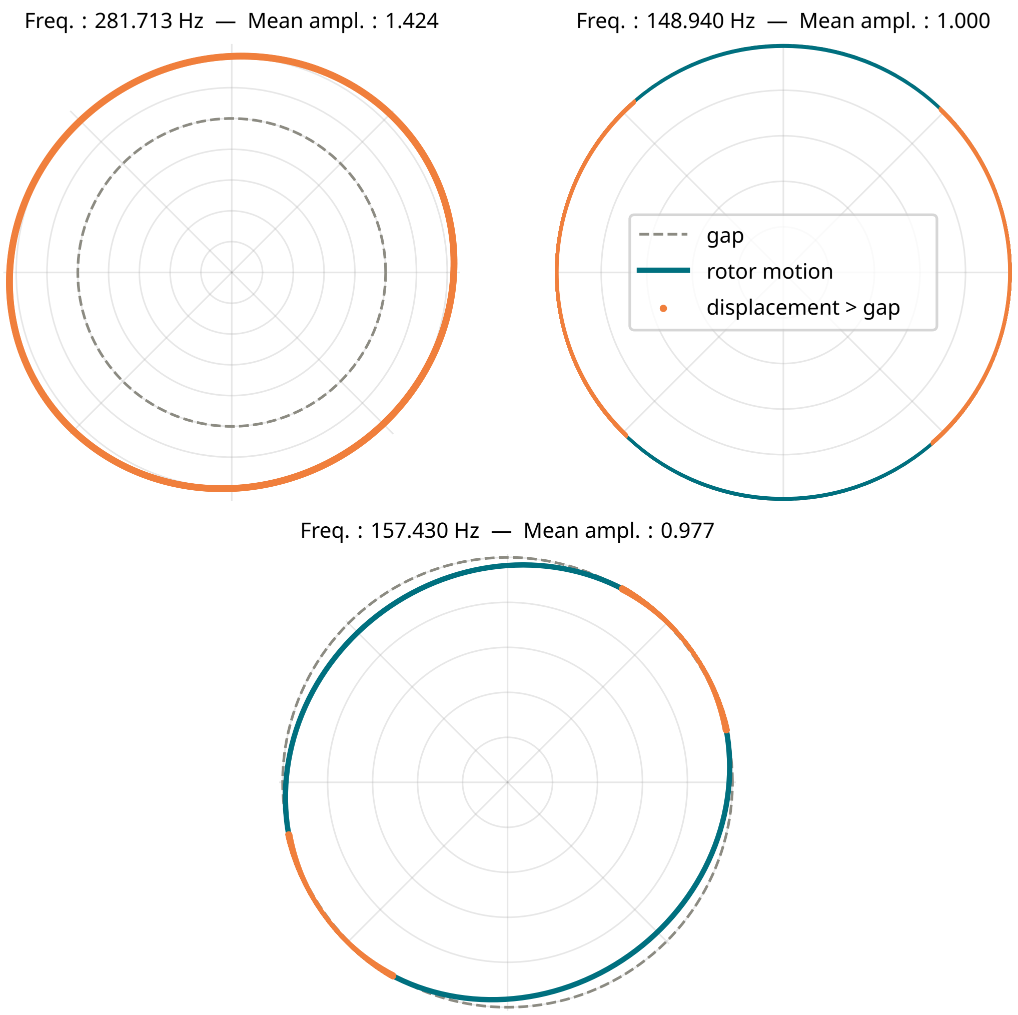


Figure 3.5 Plots of the rotor's orbital motion at the locations of the three detected bifurcation points.

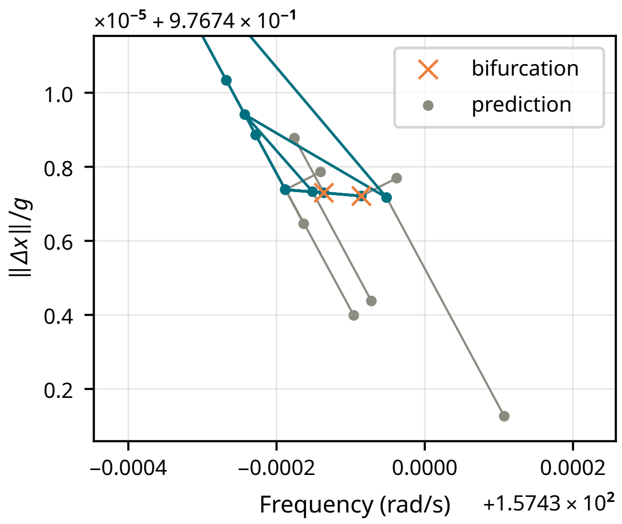


Figure 3.6 Computation of the NFRC of the asymmetric rotor model using pyHarm. This figure provides a close view of the failing region of the numerical continuation.

```
solution not accepted at om=157.42996121500374
solution converged at om=157.42984851732524
solution converged at om=157.42991446206523
solution not accepted at om=157.43002545408353
solution converged at om=157.4298118771601
solution converged at om=157.42986382636514
Warning: a limit point (fold bifurcation) was detected ⟶ path direction is reversed
solution converged at om=157.42981168164926
solution converged at om=157.429771980797
solution converged at om=157.4297322576347
```

Figure 3.7 Console log indicating the reversal of path direction at the failure point of the asymmetric model.

3.3 A First Short-Term Solution

During the investigation and testing conducted on the asymmetric rotor model, it became apparent that the continuation eventually succeeded by merely changing the input parameters of the problem. Specifically, by increasing the number of time samples n_t taken for the AFT procedure, the continuation was able to pass the problematic failure point. This successful continuation is illustrated in Figure 3.8.

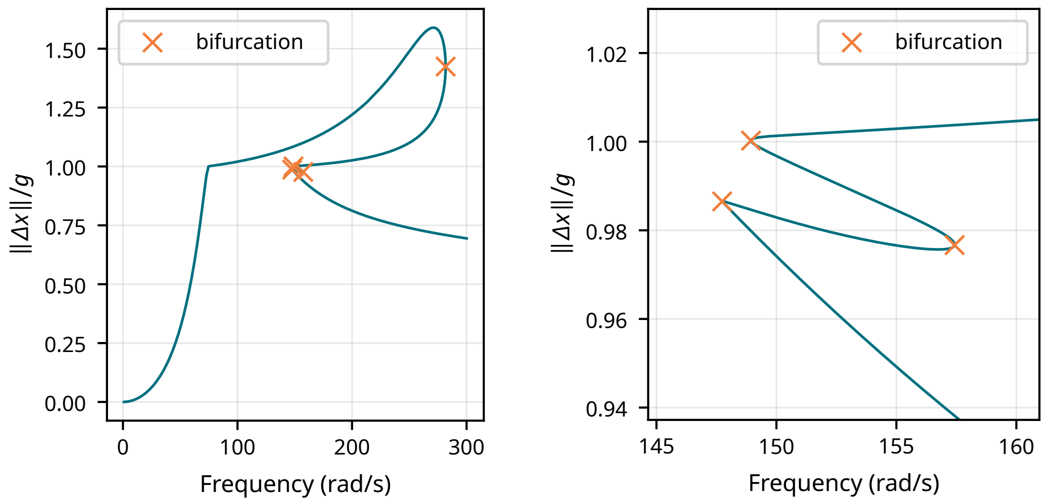


Figure 3.8 Successful continuation of the asymmetric rotor model. To pass the failing point, the initial number of 1024 time samples taken in the AFT procedure was increased to 2048. Asymmetric model.

To obtain this solution, n_t was set to 2048, while its default value was 1024. However, there is no indication that n_t was insufficient. Since only one harmonic is used in the solved HB equations, a time sample count of 1024 should be more than adequate.

A possible hypothesis is that, due to the solver's significant difficulties in maintaining proper continuation at the failure point, even a slight change in the computed solution can drastically alter the outcome of the numerical continuation. Thus, as previously stated, it appears that pyHarm is unable to properly handle the problematic bifurcation point.

The solution proposed here is not truly a solution. Merely changing input parameters and hoping that the continuation in pyHarm will not become stuck or flipped is not a sustainable long-term strategy.

In light of the analysis presented in this chapter, it has been decided that the work conducted in this thesis will primarily focus on improving the bifurcation handling capabilities of pyHarm.

This page intentionally left blank.

4 Bifurcation Handling

IMPROVING the bifurcation handling capabilities of pyHarm is the primary focus of this thesis. Efforts have been made to ensure that both fold bifurcations and branching points are properly detected, and that auxiliary solution branches can be effectively tracked.

This chapter begins by outlining the preliminary modifications made to pyHarm before implementing the bifurcation detection and branch-following techniques. Notably, a new Newton-Raphson nonlinear solver has been integrated, and the computation of the NFRC tangents has been revised.

The second section describes the modifications that are brought to the bifurcation detection mechanism, detailing how it enables seamless jumps over bifurcation points. The updated mechanism is subsequently tested on two Duffing oscillators.

In the third section, the process of branch switching is explained through the introduction of a small perturbation in the HB equations. The method is progressively refined throughout the section to enhance its usability.

Finally, the last section presents a more robust approach to branch switching. This method allows for precise identification of bifurcations and determination of the directions of the resulting secondary branches by addressing the algebraic bifurcation equation.

The methods discussed in this chapter are primarily derived from Allgower and Georg [4] and, to a lesser extent, from Seydel [17].

It is important to note that the explanations provided in this chapter focus on the theoretical underpinnings of the methods at a mathematical level. Practically, a set of new bifurcation handling objects has been developed in Python and integrated into pyHarm. The concrete implementation of these objects is detailed in Section A.2.

Reference Models — Each branch detection and switching technique presented in this chapter will be tested against two reference models from the literature. These models consist of two instances of a Duffing oscillator (Equation (1.1)), that are taken from Lazarus and Thomas [11] and Petrov [14]. They will be loosely reffered to as the *Lazarus example* and the *Petrov example* in the following. The NFRCs computed in these two papers are illustrated in Figures 4.1 and 4.2.

In the Lazarus example, the Duffing parameters are set to $m = 1\,\mathrm{kg}$, $c = 0.05\,\mathrm{N\,s/m}$, $k = 1\,\mathrm{N/m}$, $\gamma = 1\,\mathrm{N/m^3}$, and $f = 1\,\mathrm{N}$. The HB continuation is conducted with $n_h = 12$. All methods described in this chapter will be systematically tested against this model.

In the Petrov example, the Duffing parameters are set to $m = 1\,\mathrm{kg}$, $c = 1\,\mathrm{N\,s/m}$, $k = 1000\,\mathrm{N/m}$, $\gamma = 10\,000\,\mathrm{N/m^3}$, and $f = 2000\,\mathrm{N}$. The HB continuation is conducted with $n_h = 19$. When feasible, this model will serve as a validation example.

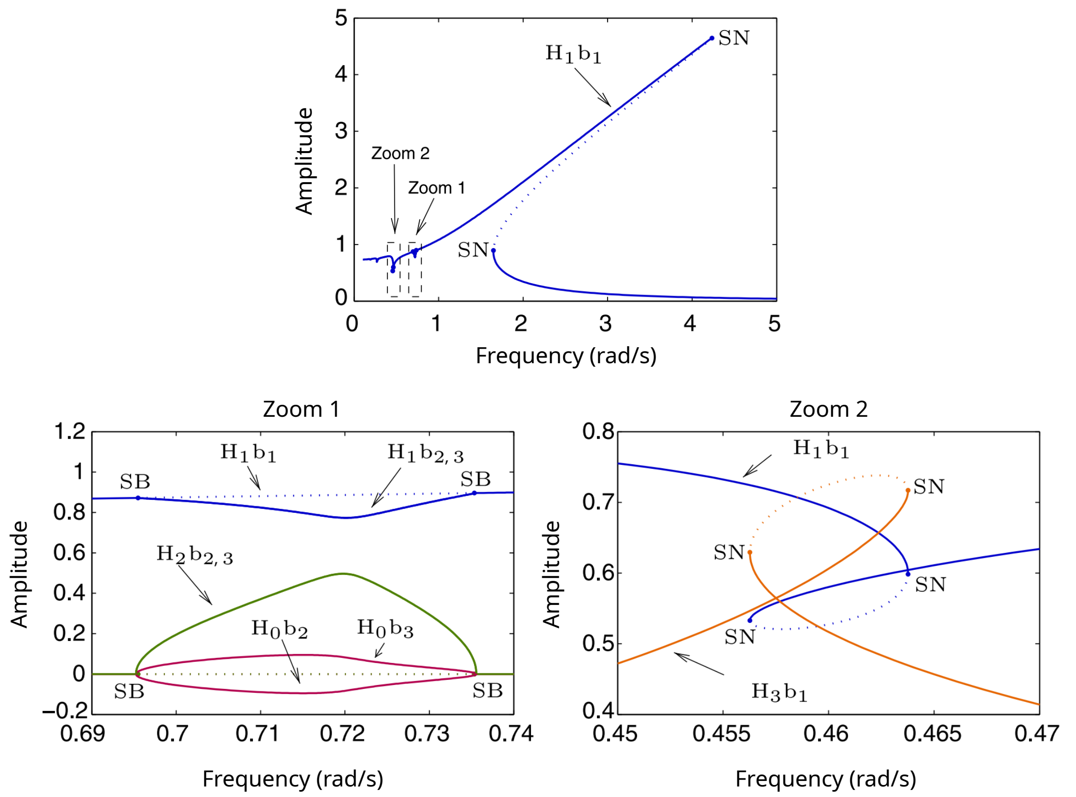


Figure 4.1 Reference Duffing oscillator from Lazarus and Thomas [11]. The notation H_xb_y denotes the y-th solution branch from the x-th harmonic component of the solution.

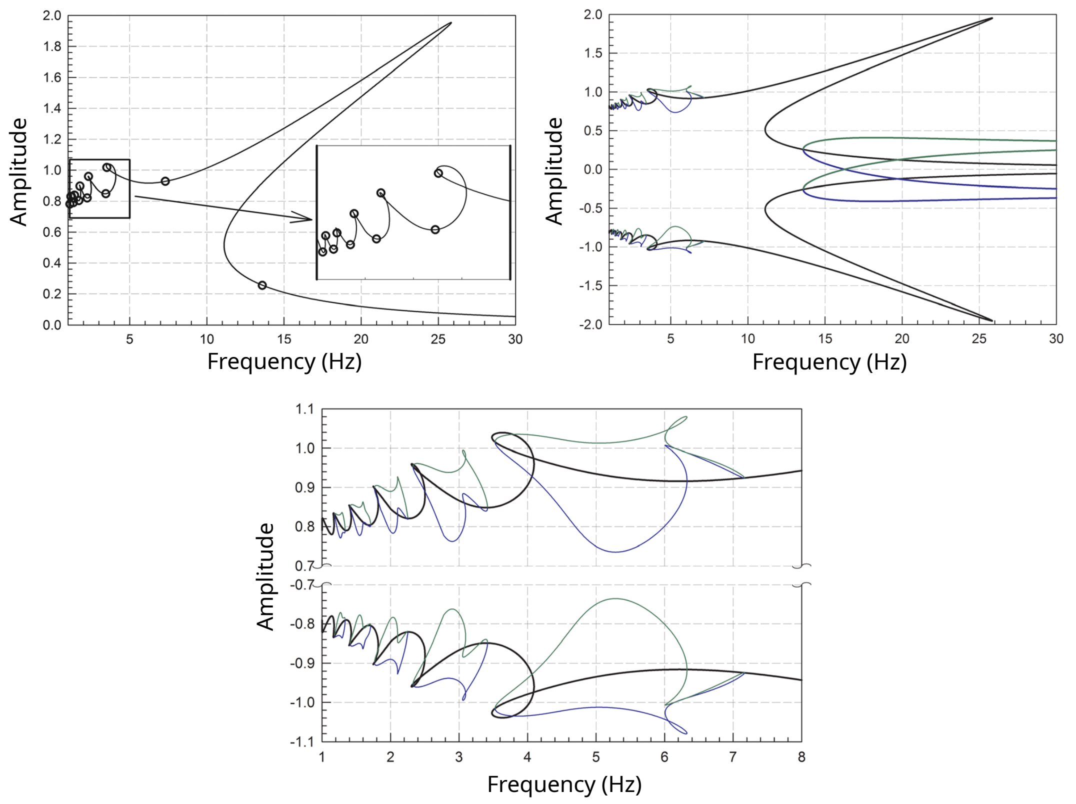


Figure 4.2 Reference Duffing oscillator from Petrov [14]. The green and blue branches represent all the secondary branches that have been detected.

4.1 Preliminary Modifications of pyHarm

The numerical continuation algorithms described in Allgower and Georg [4] are based on a series of underlying steps that require specific computations of certain quantities. For instance, the prediction step necessitates the computation of the NFRC tangent through a particular QR decomposition, while the corrections of the iterative solver are executed without a closure equation.

However, it has been observed that pyHarm computes some of these quantities in a significantly different manner. Therefore, preliminary modifications are necessary for pyHarm to ensure that these intermediate quantities are computed in accordance with the algorithms presented by Allgower and Georg.

4.1.1 Correction Step

The pyHarm codebase already includes a set of iterative solvers that can be selected for performing the correction steps. Among these is a solver based on the Newton-Raphson procedure (see Section 2.2.1). In the current version of pyHarm, this solver computes corrections based on the closed form of the problem, as detailed in Section 2.2.2. The software then allows users to choose a closure equation, such as the arc-length parametrization.

While this solver is fully functional, the techniques to be developed require modifications to the iterative solver. These modifications cannot be implemented with the current solver's design. Specifically, the use of the closure equation is too tightly coupled with the solver, making it challenging to introduce changes.

Consequently, a new Newton-Raphson solver has been developed to handle the correction steps. This new implementation no longer utilizes a closure equation. Instead, it addresses the underdetermined nonlinear system by computing corrections based on the pseudo-inverse of the Jacobian $\partial_X \hat{\boldsymbol{r}}$.

For a generic matrix $\boldsymbol{A}$ of size $(N \times N+1)$, the Moore-Penrose inverse, or pseudo-inverse, is defined as follows:

$$\boldsymbol{A}^+ = \boldsymbol{A}^*(\boldsymbol{A}\boldsymbol{A}^*)^{-1} . \tag{4.1}$$

It is important to note that for the pseudo-inverse to be defined, $\boldsymbol{A}$ must be of full rank.

With this approach, a new correction step can be designed as follows:

$$X^{(i+1)} = X^{(i)} - \partial_X \hat{\boldsymbol{r}}(X^{(i)})^+ \hat{\boldsymbol{r}}(X^{(i)}) . \tag{4.2}$$

This procedure closely resembles the Newton-Raphson iterations presented in Equation (2.26), except that the Jacobian inverse is replaced by its pseudo-inverse. It can be demonstrated [4] that this new iterative procedure effectively solves the following minimization problem:

$$\min_X \left\{ ||X_{\text{pred}} - X|| \mid \hat{\boldsymbol{r}}(X) = \boldsymbol{0} \right\} \tag{4.3}$$

where X_{pred} is the predicted solution obtained using, for example, the Euler prediction of Equation (2.30). Thus, the iterations based on the pseudo-inverse of the Jacobian find the solution of the NFRC that is closest to the prediction X_{pred}. This is illustrated in Figure 4.3.

Finally, it is important to note that the initial Newton-Raphson solver was operational.

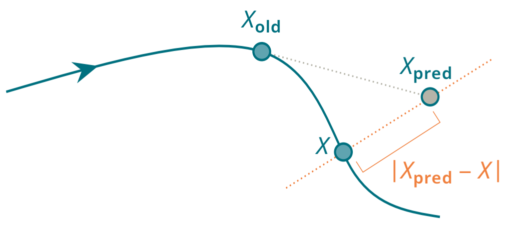


Figure 4.3 Illustration of the working principles of a pseudo-inverse-based correction step.

Anticipating future developments, the use of pseudo-inverse-based corrections will facilitate the introduction of perturbations in the solved equations.

4.1.2 Prediction Step

The tangent prediction in pyHarm has been rewritten. The software already includes a functional tangent predictor; however, aligning it more closely with the guidelines provided by Allgower and Georg will facilitate the implementation of subsequent algorithms.

The tangent is computed using a QR decomposition of the Jacobian $\partial_X \hat{\boldsymbol{r}}$:

$$\partial_X \hat{\boldsymbol{r}} = \boldsymbol{Q} \begin{pmatrix} \boldsymbol{R} \\ \boldsymbol{0}^* \end{pmatrix}, \tag{4.4}$$

where $\boldsymbol{Q}$ is an orthogonal matrix of size $n_q(2n_h+1)+1$, $\boldsymbol{R}$ is an upper triangular matrix of size $n_q(2n_h+1)$, and $\boldsymbol{0}$ is a zero column vector. It can then be shown [4] that:

$$\det \begin{pmatrix} \partial_X \hat{\boldsymbol{r}} \\ \boldsymbol{z}^* \end{pmatrix} = \det \boldsymbol{Q} \, \det \boldsymbol{R}, \tag{4.5}$$

where $\boldsymbol{z}$ denotes the last column of $\boldsymbol{Q}$. Referring to the definition of the tangent provided in Equation (2.32), it can be observed that the vector $\boldsymbol{z}$ corresponds to the desired tangent, up to a sign. In other words, $\boldsymbol{t}(\partial_X \hat{\boldsymbol{r}}) = \pm \boldsymbol{z}$. According to the tangent definition, the components of Equation (4.5) must be positive, indicating that the sign to be applied to the computed $\boldsymbol{z}$ is determined by $\operatorname{sign}(\det \boldsymbol{Q} \, \det \boldsymbol{R})$. Note that, since $\boldsymbol{R}$ is an upper triangular matrix, its determinant can be easily computed as the product of its diagonal elements.

The newly implemented predictor operates in two steps. First, the tangent is determined at the last computed solution of the NFRC. Second, the next predicted solution is computed using an Euler prediction, effectively implementing Equation (2.30).

4.2 Jumping Over a Bifurcation

The most straightforward approach to handling bifurcations is to avoid addressing them directly. Therefore, this section outlines how bifurcations can be effectively detected and how to bypass them.

Currently, pyHarm provides a mechanism for detecting and passing over branching points and other types of bifurcation points, aside from simple folds. However, this mechanism has two significant limitations. First, it does not label the detected bifurcations according to the classification outlined in Section 2.3; the code only indicates the presence of folds. Second, the method used to update the tangent direction of the NFRC is incompatible with the branch-switching algorithms developed in subsequent sections.

Consequently, this section begins to describe the rewritten bifurcation detection and jump mechanism in pyHarm. After that, this new implementation will be tested on the two reference Duffing oscillators shown in Figures 4.1 and 4.2.

4.2.1 New Detection and Jump Mechanism

To clarify the changes made to the pyHarm code, refer to Figure 4.4, which compares the working principles of the bifurcation detection and jump mechanism in both the original and modified code.

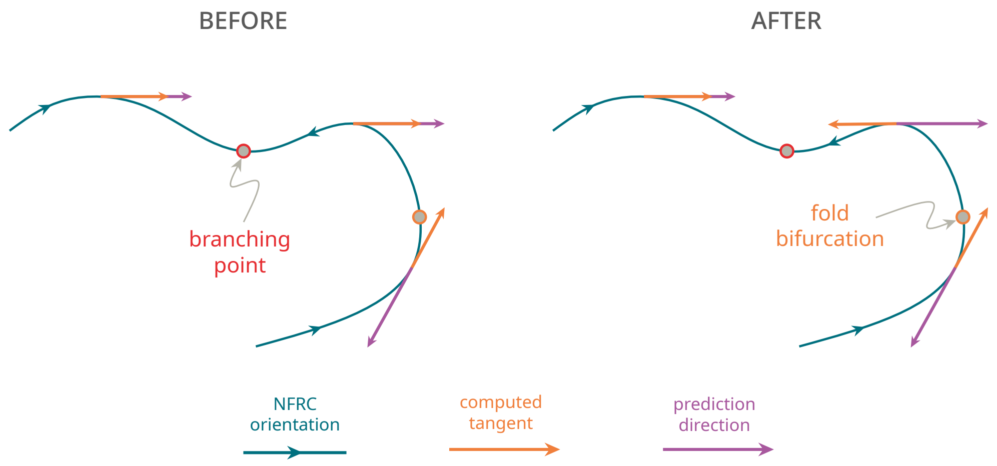


Figure 4.4 Working principles of the bifurcation detection and jump mechanism in the original (BEFORE) and modified (AFTER) code of pyHarm.

The original code computes the tangents of the NFRC such that they always point towards increasing frequencies, effectively directing them ”to the right” on a frequency response graph. Specifically, the tangent is initially computed in a manner consistent with the explanation in Section 4.1.2. At this stage, the computed tangent aligns with the orientation of the NFRC. However, the code then immediately multiplies these tangents by the sign of their Ω-component, which results in a consistent direction for the tangents. Consequently, passing through a branching point or any higher-order bifurcation does not alter the tangent direction.

To bypass a fold bifurcation, the code must be able to reverse the search direction for the next prediction. To achieve this, pyHarm distinguishes between the tangent of the curve and the direction of prediction. Internally, the Euler prediction in Equation (4.6) is performed as follows:

$$X_{\text{pred}}^{(i+1)} = X^{(i)} + h\,\sigma\,\tilde{\boldsymbol{t}}(\partial_X \hat{\boldsymbol{r}}(X^{(i)}))\,. \tag{4.6}$$

The tangent is denoted by $\tilde{\boldsymbol{t}}$ to indicate that it does not correspond to the tangent $\boldsymbol{t}$ defined in Equation (2.32). The variable σ takes values of ± 1. The original pyHarm code flips the value of σ when a fold bifurcation is detected, using the criterion provided in Equation (2.33). Thus, what is referred to as the prediction direction corresponds to $\sigma\,\tilde{\boldsymbol{t}}$.

This mechanism has the advantage of robustness; regardless of the type of bifurcation detected—branching points or others—the continuation maintains a consistent direction without the need for precise identification of these bifurcations. The code only needs to correctly detect simple folds to function properly and navigate around them.

In contrast, the modified version of the code computes the tangents of the NFRC curve in accordance with the definition in Equation (2.32). This means that the tangents correspond to the defined orientation of the NFRC. According to Equation (2.34), the tangent changes sign when encountering a branching point, following the criterion outlined in Equation (2.34). To maintain a consistent prediction direction, the variable σ is now flipped when a branching point is detected. The predicted direction is then given by $\sigma \boldsymbol{t}$.

In addition to branching points, the modified code also flips σ when the tangent experiences abrupt direction changes, without necessarily detecting a branch point. It is indeed possible for the numerical continuation to encounter a bifurcation that is neither a fold nor a branching point, which alters the direction of the NFRC. To address this, a simple criterion has been added to the detection mechanism: σ will flip if the computed tangent is significantly misaligned with the direction line drawn from the two most recent solutions. This is illustrated in Figure 4.5, where the "blind cone" defines the region within which a predicted tangent will be considered reversed due to a potential bifurcation. In practice, the user can now specify an angle value for the cone's aperture.

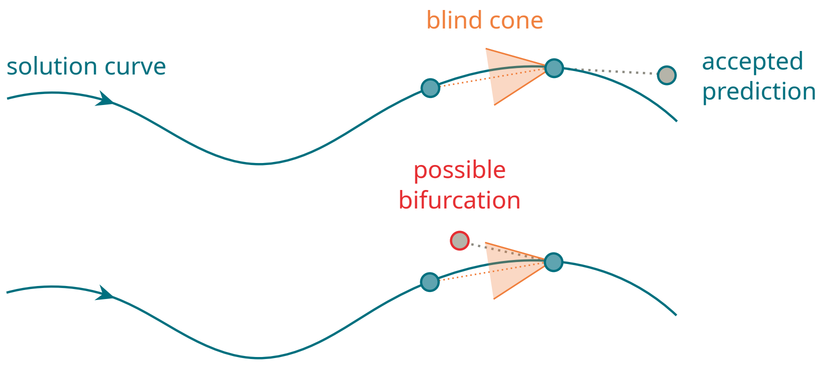


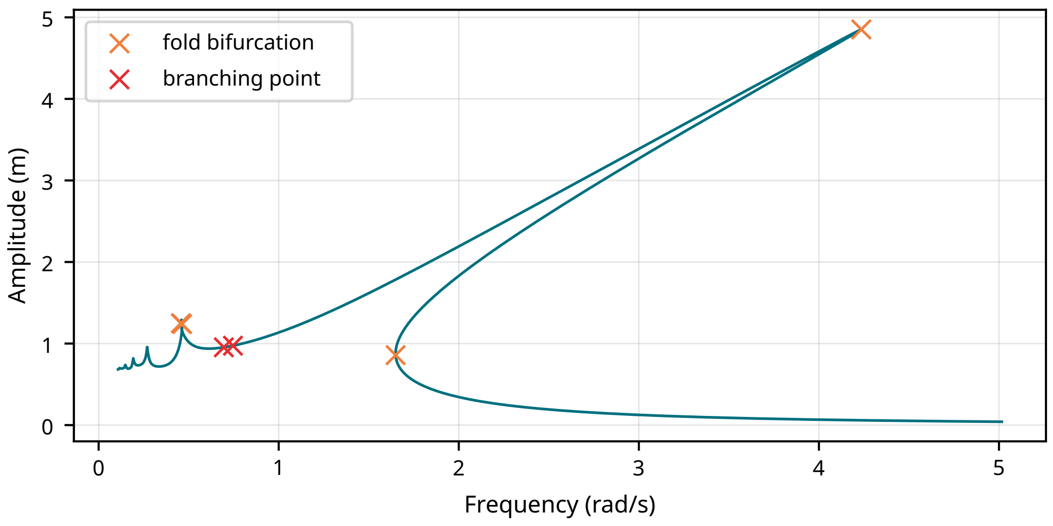
Figure 4.5 Illustration of the working principle of the blind cone, designed to improve the robustness of the continuation process.

It is important to note that the NFRC presented in the graphs and schematics of this thesis are depicted in two dimensions. The solutions of the HB equations are indeed reduced to a single scalar, which is plotted against the excitation frequency. In the context of structural dynamics, this scalar typically represents the amplitude of motion at a selected DOF. Therefore, the angle computed by the blind cone criterion is in fact determined from the scalar product between two vectors in the $\mathbb{R}^{n_q(2n_h+1)+1}$ space.

4.2.2 Validation Against Reference Articles

To validate the new implementation of the bifurcation detection and jump mechanism, the modified code is tested against the two reference Duffing oscillators shown in Figures 4.1 and 4.2.

Results are displayed in Figures 4.6 and 4.7. Overall, it can be assessed that both the fold bifurcations and the branching points have been correctly detected.



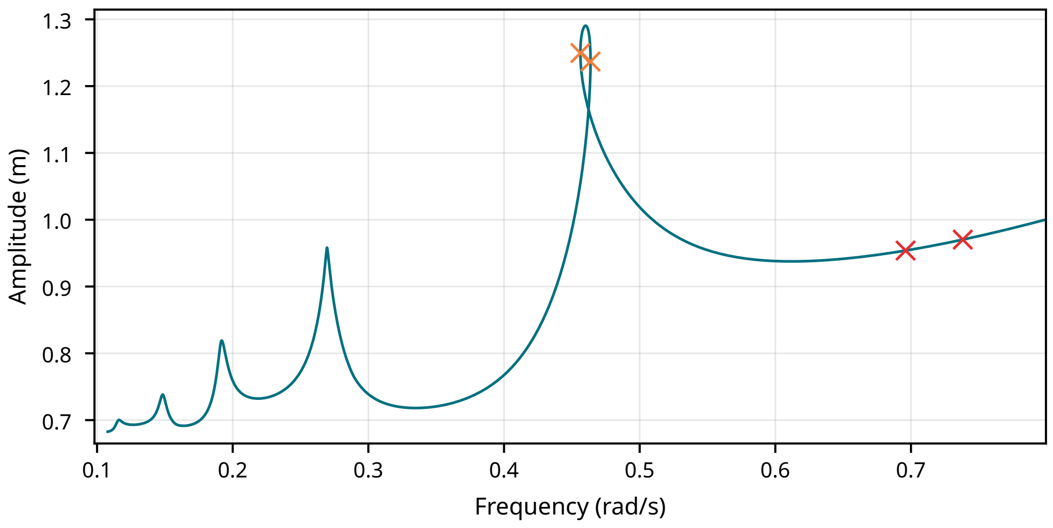


Figure 4.6 Application of the new bifurcation detection and jump mechanism to the Lazarus example. The second graph provides a closer view of the first one.

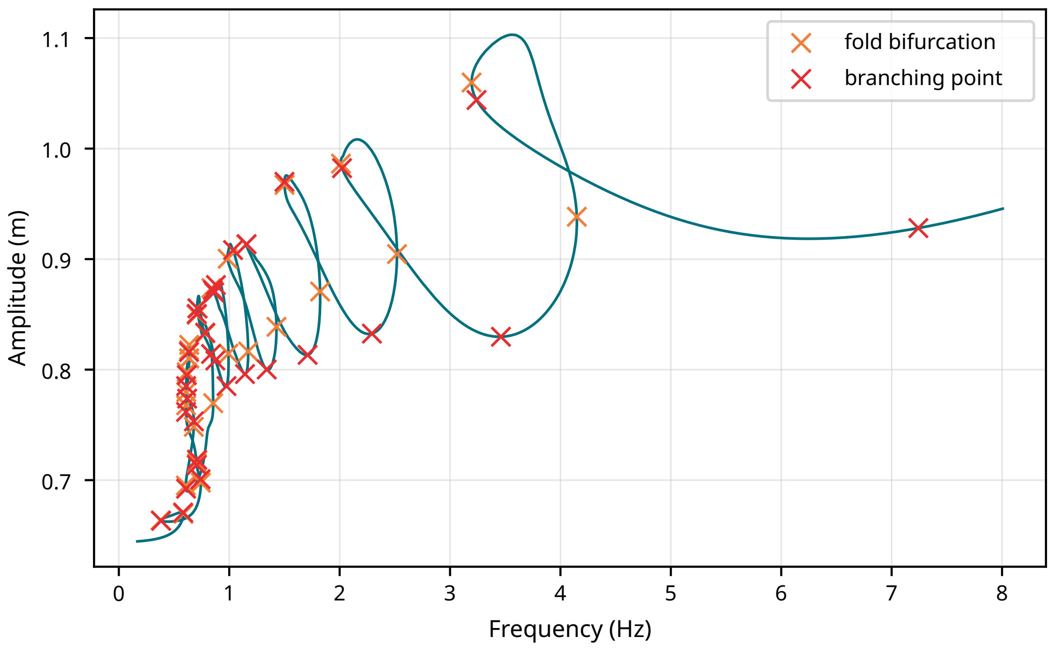


Figure 4.7 Application of the new bifurcation detection and jump mechanism to the Petrov example.

In the Lazarus example, two branching points have been identified. By comparing with the reference graphs, it appears that these points are correctly located and correspond to the positions where the alternative branches of the zeroth, first, and second harmonics emerge.

A significant number of bifurcations are detected within the restricted frequency range of 0 Hz to 8 Hz for the Petrov example. The folds for each loop are accurately detected, and the branching points align well with the emergence locations of the auxiliary branches.

Finally, it is noteworthy that no tangent flips were triggered by the blind cone condition in these two examples.

4.3 Switching Branches *via* Perturbation

This section presents a first method for switching from the main solution branch to an auxiliary one following the detection of a branching point. This method involves introducing a small perturbation into the harmonic balance equations, resulting in what are referred to as the *perturbed equations*.

The section begins by explaining the overall principle behind the method of small perturbation. It is then explained how perturbations can be applied globally to the HB equations. Subsequently, a more refined technique is introduced, where perturbations are applied locally at the detected branch point. Finally, it is demonstrated how this latter technique can be made interactive, allowing the user to introduce a desired perturbation during the ongoing numerical continuation.

4.3.1 General Method Overview

The method of small perturbation is one of the simplest to implement numerically. Its operation is based on the Sard theorem [4]. One consequence of this theorem is that when a small perturbation $\boldsymbol{\epsilon} \in \mathbb{R}^{n_q(2n_h+1)+1}$ is randomly chosen, then the curve implicitly defined by $\hat{\boldsymbol{r}}(\boldsymbol{\epsilon})^{-1}$ will not have any bifurcation points.

This implies that a numerical continuation similar to that described in Equation (2.27) can be employed, with the HB equations replaced by the perturbed equations:

$$\hat{\boldsymbol{r}}(X) = \boldsymbol{\epsilon}\,. \tag{4.7}$$

This approach enables tracking a curve in the $\mathbb{R}^{n_q(2n_h+1)+1}$ space that is close to the solution curve of the HB equations but does not exhibit bifurcations. This is illustrated in Figure 4.8.

If the perturbation is chosen to be *sufficiently small*, solving Equation (4.7) can yield a reasonable approximation of the solution to the HB equations. As illustrated in Figure 4.8, the perturbed solution curve will not pass through the bifurcation but will instead circumvent it by following a secondary branch. Following this perturbed solution curve thus provides a method for switching from the main solution branch.

The challenge of this method lies in selecting the appropriate perturbation value. Indeed, “sufficiently small” is somewhat ambiguous. A perturbation that is too small will cause the continuation to remain on the main branch, resulting in the secondary branch of interest becoming indistinguishable from the main branch. Generally, as the

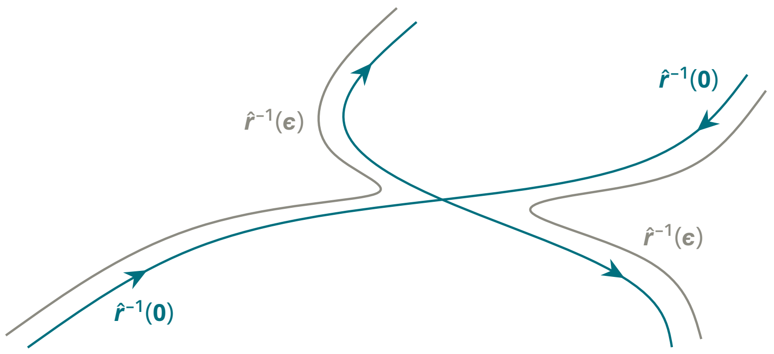


Figure 4.8 Illustration of a NFRC in the vicinity of a branching point. The adjacent grey curve, obtained by introducing a perturbation ϵ into the HB equations, does not pass through the bifurcation. Drawing reproduced and adapted from Allgower and Georg [4].

perturbation level decreases, the step size h must also be reduced to accurately track the perturbed curve. Conversely, if the perturbation is too large, the perturbed curve will deviate too far from the desired solution, rendering the solution of the perturbed equations ineffective.

Thus, there is a trade-off in choosing the value for ϵ. It is difficult to provide general recommendations for default values, as this depends on the specific problem being solved. As a preliminary estimate, Allgower and Georg [4] suggests starting with $\epsilon = 10^{-4}$.

4.3.2 Introducing a Global Perturbation

The most straightforward way to implement the small perturbation method is to directly modify the nonlinear solver used for numerical continuation.

In this context, the refactoring of the Newton-Raphson solver discussed in Section 4.1.1 proves useful. With the newly written solver, it becomes straightforward to introduce a small perturbation into the HB equations to be solved. The pseudo-inverse-based correction step described in Equation (4.2) can be modified to:

$$X^{(i+1)} = X^{(i)} - \partial_X \hat{\boldsymbol{r}}(X^{(i)})^{+}(\hat{\boldsymbol{r}}(X^{(i)}) - \epsilon)\,. \tag{4.8}$$

This iterative procedure now corresponds to solving the following minimization problem:

$$\min_X \left\{ ||X_{\text{pred}} - X|| \mid \hat{\boldsymbol{r}}(X) = \epsilon \right\}\,. \tag{4.9}$$

By making this change directly at the solver level, the computation of the perturbed equations will be performed throughout the entire continuation procedure.

From the user's perspective, it is now possible to set a custom perturbation value for the solver before pyHarm begins the numerical continuation. According to the aforementioned recommendation by Allgower and Georg, this perturbation is set by default to $\epsilon = 10^{-4}$.

Validation Against Reference Article

The previously described technique has been applied to the Duffing oscillator from Lazarus and Thomas. Results are shown in Figure 4.9. The curve presented here corresponds to the solution of the perturbed equation, solved with $\epsilon = 10^{-4}$. This curve can be compared with the NFRC obtained in Figure 4.6.

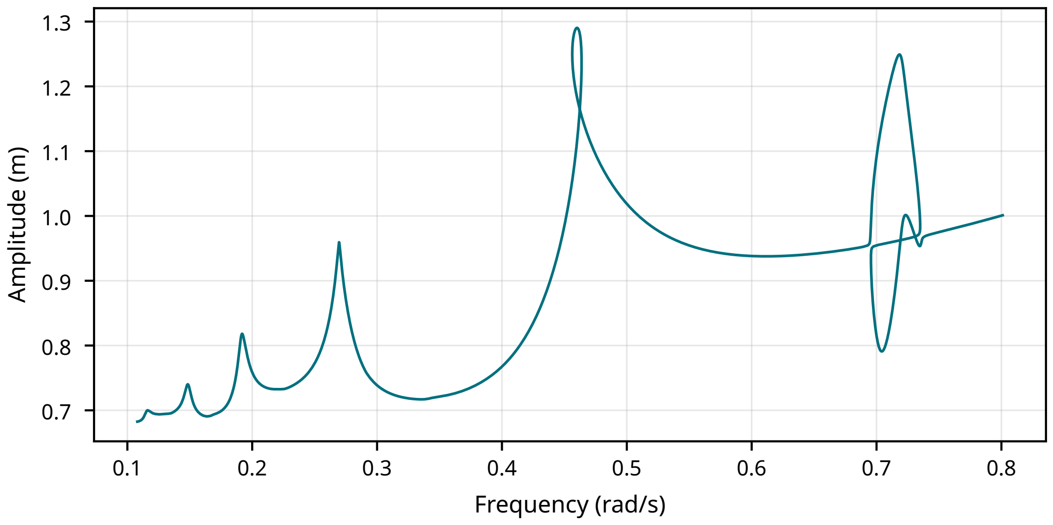


Figure 4.9 Perturbed solution curve in the frequency range from 0.1 rad/s to 0.8 rad/s for the Lazarus example. A global perturbation of $\epsilon = 10^{-4}$ is applied.

In the range 0.1 rad/s to 0.6 rad/s, the perturbed solution curve approximates the NFRC reasonably well, suggesting that the chosen perturbation value is sufficiently small.

In the branching region, the perturbed solution curve successfully branches off from the main NFRC. Notably, multiple auxiliary branches have been identified. Upon closer examination of the graph, it is evident that the perturbed solution curve forms a continuous trajectory. As it approaches the first branching point, the continuation diverges to the upper secondary branch. However, when nearing the reattachment point (the second branching point) of this secondary branch, the perturbed solution curve does not reattach to the main branch; instead, it continues to explore additional secondary branches. After traversing several of these newly discovered secondary branches, the continuation eventually succeeds in reattaching to the main branch.

A closer view of the perturbed solution curve is provided in Figure 4.10. This graph focuses on a small region centered around the first branching point.

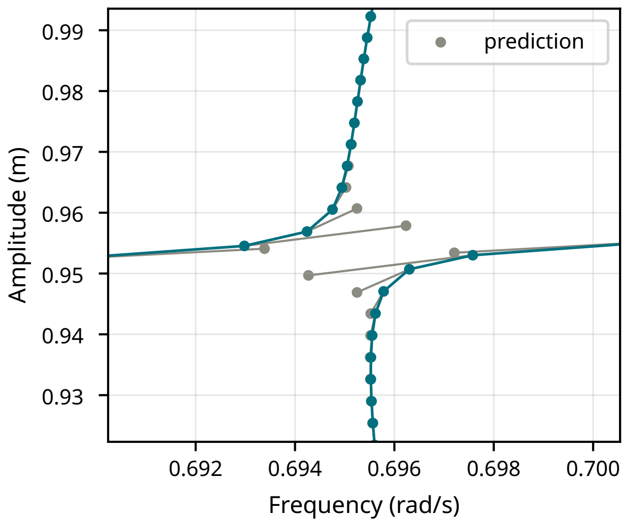


Figure 4.10 Close view of the perturbed solution curve around the first branching point of Lazarus example. A global perturbation of $\epsilon = 10^{-4}$ is applied.

This figure should be compared with the general schematic in Figure 4.8. As observed, the perturbed curve avoids passing through the branching point and is deviated upward toward a secondary branch. It is worth noting that setting a negative value for ϵ would have caused the perturbed solution curve to deviate downward.

Figures 4.9 and 4.10 present the perturbed solution curve of the Duffing oscillator from Lazarus and Thomas, taking into account all twelve first harmonics for the computation of the DOF amplitude. However, the NFRC presented in the reference paper, shown in Figure 4.1, has been filtered to include only the first harmonic components. To provide a direct comparison with these results, the computed perturbed solution curve can also be filtered. Results are shown in Figure 4.11. Details on how to achieve this filtering are explained in Section B.1.2.

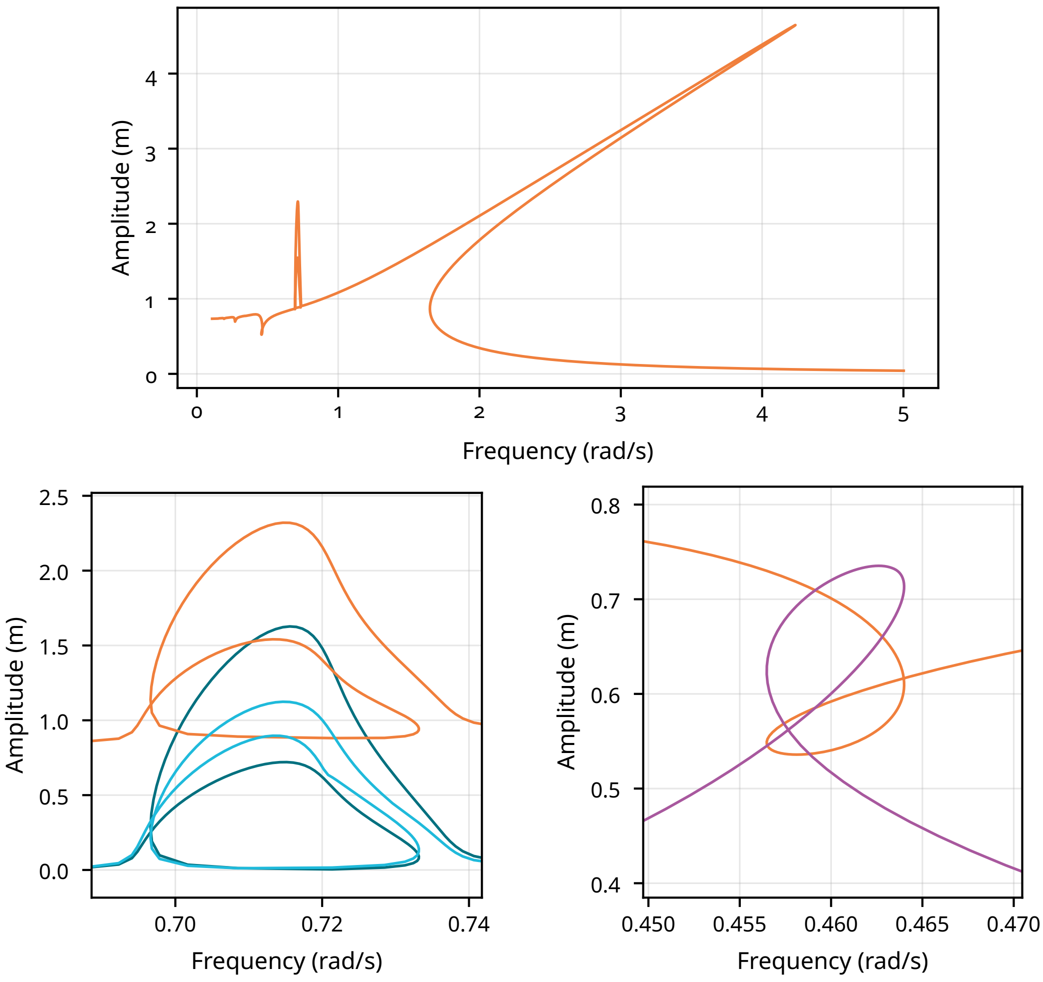


Figure 4.11 Perturbed solution curve of the Duffing oscillator from Lazarus and Thomas. A global perturbation of $\epsilon = 10^{-4}$ is applied. The harmonics have been filtered in the same manner as for the reference article results, presented in Figure 4.1. Legend: — H_0, — H_1, — H_2, — H_3.

It can be observed that the secondary branches discovered are significantly different from those in the reference paper.

Regarding the first harmonic component, the branch labeled as $H_1 b_{2,3}$ appears to have been properly identified. However, the perturbed solution curve reveals the existence of two additional secondary branches that are not present in the reference article results. For the zeroth and second harmonic components, the symmetry-breaking branches noted in the reference article are not retrieved by the perturbed solution curve. Instead, branches with much larger amplitudes are observed.

In Summary — The results obtained from this initial implementation of the small perturbation technique highlight a significant drawback of the method. There is a clear lack of control regarding the selection of solution branches to traverse. Changing the perturbation value ϵ can completely alter the type and/or order of the secondary branches explored by the numerical continuation. While this method can be useful for exploring the auxiliary branches of a system by adjusting the perturbation parameter ϵ, it does not effectively isolate a desired solution branch.

4.3.3 Introducing a Local Perturbation

The previous section highlighted that introducing a global perturbation to the HB equations constitutes a branching-off method that lacks control in selecting a specific branch to follow. Furthermore, it is important to note that the perturbed solution curve does not correspond to the actual NFRC of the system. Therefore, it is desirable to retrieve the solution of the unperturbed HB equations when a perturbation is unnecessary, specifically outside the vicinity of a branching point.

To address this issue, a second, more refined perturbation method is proposed. In this approach, the perturbed equations are solved only in the vicinity of the branching point, while the remainder of the continuation is conducted using the unperturbed equations.

Although this change may appear trivial, its implementation is not straightforward. With the global perturbation method, it was sufficient to input a perturbation value into the pyHarm entry data. This action set the solver to the desired perturbation value before the numerical continuation begins. Then the entire continuation is performed with that perturbation.

In contrast, the current method requires the perturbation parameter ϵ to be adjusted during the continuation process. Due to the internal structure of pyHarm, this necessitated a comprehensive revision of the general continuation procedure. This revision led to the development of a separate abstract class to implement all the bifurcation techniques discussed in this chapter. Once again, for practical details, please refer to Section A.2.

The modifications made to pyHarm now allow for the adjustment of the perturbation parameter during the continuation process. This capability enables the implementation of a local perturbation method, the principle of which is illustrated in Figure 4.12.

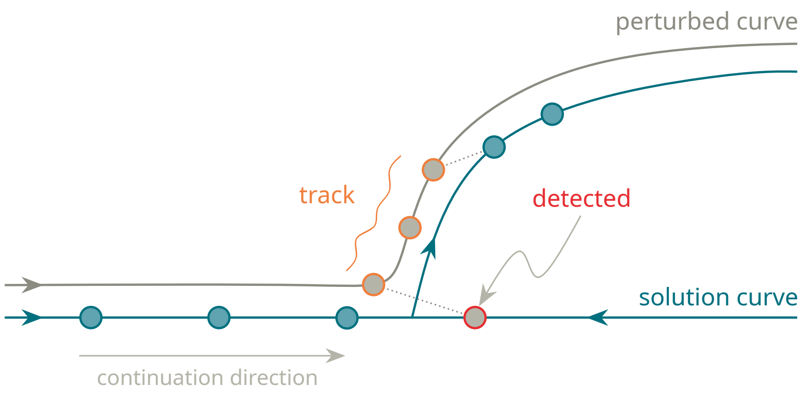


Figure 4.12 Illustration of the local perturbation method.

As depicted, the continuation is conducted using the unperturbed HB equations until a branching point is detected. Upon detection, the prediction direction is not reversed, as would typically occur, as shown in Figure 4.4. Instead, the next prediction is sought in the backward direction of the continuation. At this point, a perturbation is introduced into the system to solve. The nonlinear solver then executes the correction steps as outlined in Equation (4.8). Consequently, the next solution of the continuation is identified prior to the bifurcation point, along the perturbed solution curve. The subsequent solutions of the continuation are computed from the perturbed equations. After a specified number of continuation steps, the solver reverts to the unperturbed equations, and the newly computed solutions are reattached to the sought secondary branch of the NFRC.

It is important to note that this local perturbation method is also activated when a tangent flip is detected, specifically when the blind cone criterion illustrated in Figure 4.5 is not satisfied.

From a user perspective, the pyHarm software now allows for the setting of the perturbation parameter value, as before, along with the specification of the number of continuation steps n_s to be performed on the perturbed solution curve. The default values are set to $\epsilon = 10^{-4}$ and $n_s = 5$.

Validation Against Reference Article

The algorithm described has been applied to the Lazarus example, with results presented in Figure 4.13.

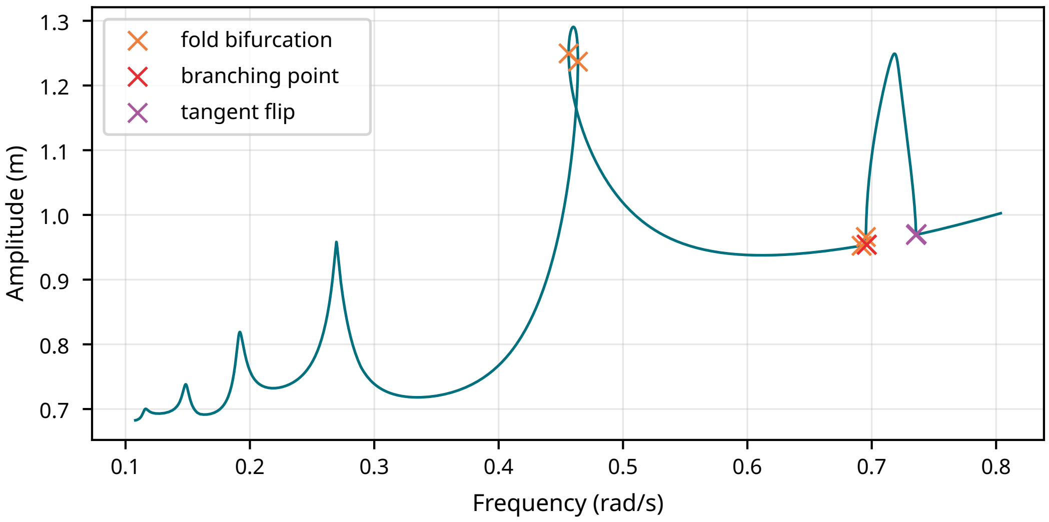


Figure 4.13 Application of the local perturbation method on the Lazarus example. The local perturbation parameter has been set to -10^{-5}.

As discussed in Figure 4.9, setting a global perturbation to 10^{-4} yielded satisfactory results outside the branching regions, with the perturbed solution closely resembling the unperturbed one. Therefore, while retrieving the unperturbed solution outside the branching region is beneficial, the local perturbation method does not significantly enhance the computed results in practice.

It is evident that a unique solution branch has been extracted in this instance. However, this should not be interpreted as a definitive improvement afforded by the new local perturbation method. In fact, setting a perturbation value of $\epsilon = -10^{-5}$ fortuitously aligns with the extraction of one of the secondary branches.

Figure 4.14 provides a detailed view of the same continuation solution shown in Figure 4.13.

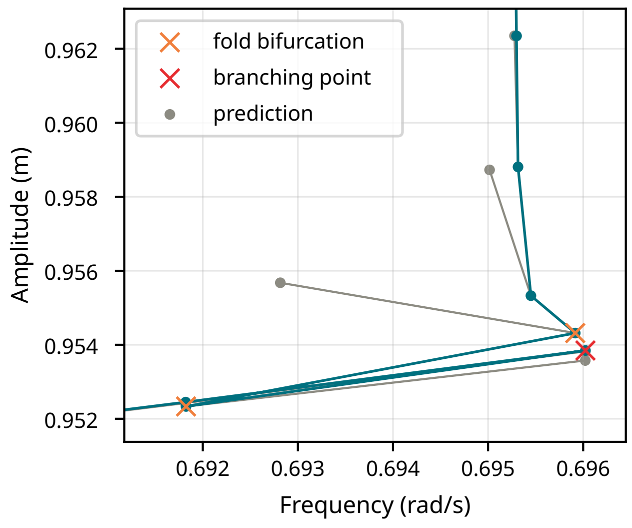


Figure 4.14 Close view of the local perturbation method applied to the Lazarus example. The local perturbation parameter has been set to -10^{-5}.

This figure illustrates the computations performed by the local perturbation algorithm near the first branching point. It is beneficial to compare this figure with the theoretical schematic in Figure 4.12. It can be observed that when the branching point is detected, the continuation indeed computes a solution in the backward direction, which is slightly offset from the main branch of the NFRC and lies on the perturbed solution curve. The subsequent solutions are then computed along this perturbed curve.

Additionally, it is noteworthy that some fold bifurcations are detected. However, these are artifacts generated by the local perturbation algorithm and do not represent physical fold bifurcations.

In Summary — The local perturbation method does not significantly enhance the computed solution on the main NFRC branch, nor does it improve the controllability of the selected secondary branch. However, this method will demonstrate its utility when employed interactively, as described in the following section.

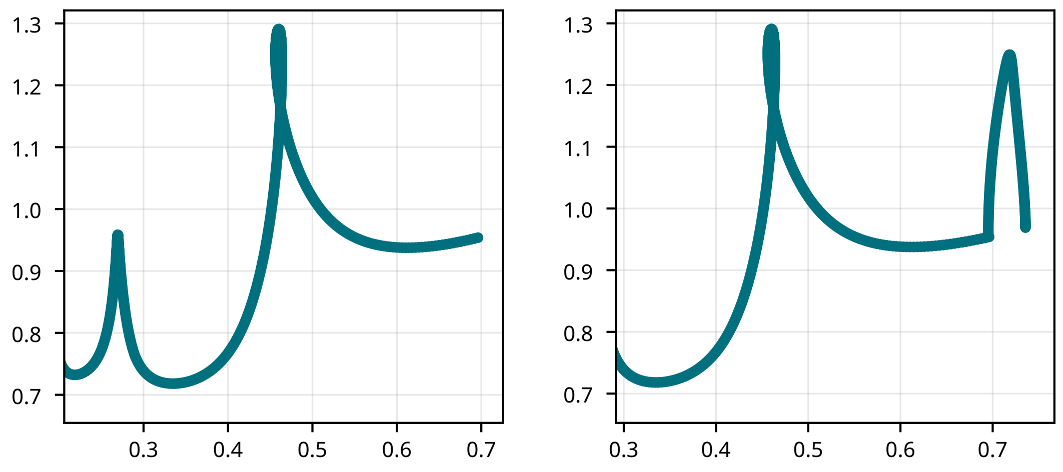
Interactive Branch Selection

The primary limitation of the perturbation-based methods developed thus far is that once the continuation procedure is initiated, there is no mechanism to control which branch the perturbed solution curve will follow. The user can only set two parameters, ϵ and n_{steps}, prior to launching the continuation procedure.

To effectively select and isolate a specific solution branch, it would be advantageous to modify the perturbation parameter interactively during the ongoing continuation. In this context, the local perturbation method has been made interactive.

Now, when a branching point (or a tangent flip) is detected during the continuation, the code halts the solving process, opens a window displaying the current state of the continuation, and offers the user the option to either follow the main NFRC branch or to introduce a custom perturbation value. This interactive functionality of pyHarm is illustrated in Section 4.3.3.

It is important to note that the two graphs are directly extracted from the pyHarm continuation execution. Consequently, no quantities or units are associated with the



```
...
solution converged at om=0.6918204312361893
solution converged at om=0.6960238287388201
Warning: a branching point bifurcation was detected
Follow [m]ain branch or explore [a]lternative branch ? a
Input perturbation (float): -1E-5
solution converged at om=0.6918237066699394
Warning: a fold bifurcation was detected
solution converged at om=0.6959164860329472
Warning: a fold bifurcation was detected
solution converged at om=0.6954504857438039
solution converged at om=0.6953149808505742
...
```

Figure 4.15 Example of an interactive use of pyHarm, on the Lazarus example. The graphs shown here are the one that appears during the solving, at two different branching detected. The console on the bottom illustrate the choice that is given to the user.

axes of these graphs, as they depend on the specific nature of the simulated problem, which is not known by pyHarm.

When the main branch option is selected, the code reverts to a classical bifurcation jump, as described in Section 4.2. Alternatively, if a value for the perturbation parameter is inputted, the local continuation algorithm is resumed.

By adjusting the perturbation parameter interactively, it is now possible to extract a single solution branch. For instance, in the interactive continuation shown in Section 4.3.3, the user inputted a perturbation value of 10^{-5} twice, which successfully extracted the upper solution branch previously identified in Figure 4.13.

Through trial and error with the values of ϵ interactively passed to the program, it is feasible to extract other solution branches. However, this approach is not particularly user-friendly. It can become tedious to determine the correct combination of perturbation parameters to introduce during the numerical continuation in order to extract the desired solution branch. Acknowledging this limitation, the next section will present a new method, not based on perturbations, that will enable a more reliable branch selection.

4.4 Switching Branches *via* the Bifurcation Equation

The lack of controllability associated with the perturbation methods discussed in the preceding section has prompted the development of a second, more reliable branch switching technique. This section outlines this new approach. In this method, the branching point is localized more accurately, enabling the computation of the various directions taken by the emanating branches. One of these directions can then be selected to determine the next predicted solution, effectively resuming the continuation along the chosen branch.

The section begins by detailing the global working principles and the mathematical foundations of this branch switching technique. Once again, for implementation details, the reader is encouraged to refer to Section A.2.

4.4.1 General Method Overview

The technique developed in this section will be referred to as the *branch tangents method*. The working principle of this technique can be decomposed into three main steps: detection of a branching point, precise localization of the bifurcation point, and estimation of the tangents of the emanating solution branches. These three steps are illustrated in Figure 4.16.

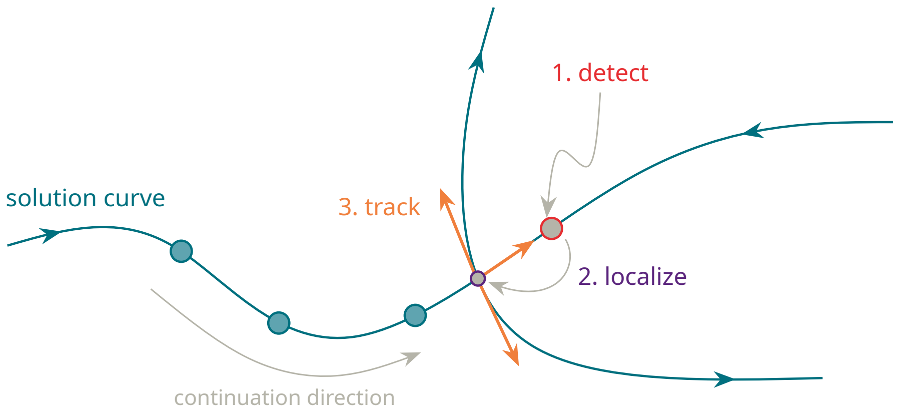


Figure 4.16 Illustration of the working principle of the branch tangents method.

1. Detect — In the first step of the branch tangents method, bifurcation detection is performed in the same manner as described in Section 4.2.1. However, in this case, the sign of the next prediction direction is not flipped (σ remains unchanged).

2. Localize — The second step of the algorithm provides a more precise location of the bifurcation point through a secant search. The definition of a branching point provided in Equation (2.34) is particularly useful here. The underlying detection criterion can be succinctly rewritten as:

$$\phi_\mathrm{b}(X) = \begin{pmatrix} \partial_X \hat{\boldsymbol{r}}(X) \\ t * (\partial_X \hat{\boldsymbol{r}}(X)) \end{pmatrix} . \tag{4.10}$$

A small secant search procedure can then be established based on this criterion, expressed as:

$$h = -\frac{\phi_\mathrm{b}(X^{(i)})}{\phi_\mathrm{b}(X^{(i-1)}) - \phi_\mathrm{b}(X^{(i)})}\, h\,. \tag{4.11}$$

More specifically, this secant search iteratively updates the value of the step size h. The secant search begins from the solution point where the bifurcation has been detected, and the preceding solution. The value of the step size is updated according to Equation (4.11). A new solution is then predicted using this updated step size, followed by classical correction steps to make this prediction converge. This procedure is iteratively applied to the two most recently computed solution points until the step size falls below a specified tolerance.

3. Track — After the bifurcation has been accurately localized, the subsequent computation of the tangents of the emanating branches is performed in multiple steps.

First, an approximation of the kernels of the Jacobians $\partial_X \hat{\boldsymbol{r}}$ and $(\partial_X \hat{\boldsymbol{r}})^*$ must be obtained at the location of the localized bifurcation, denoted as $\tilde{X}$. These kernels are represented as follows:

$$\ker \partial_X \hat{\boldsymbol{r}} = \mathrm{span}\{\boldsymbol{\tau}_1, \boldsymbol{\tau}_2\} \quad \text{and} \quad \ker(\partial_X \hat{\boldsymbol{r}})^* = \mathrm{span}\{\boldsymbol{e}\}\,. \tag{4.12}$$

To achieve this, the pre-existing `null_space()` function from the SciPy Python library is utilized.

From this point, it can be demonstrated that the tangents of the emanating branches can be determined from the solutions of the algebraic bifurcation equation, expressed as:

$$\alpha_{1,1}\xi_1^2 + 2\alpha_{1,2}\xi_1\xi_2 + \alpha_{2,2}\xi_2^2 = 0 \tag{4.13}$$

where

$$\begin{aligned} \alpha_{i,j} &= \partial_i \partial_j\, g(0,0) \\ g(\xi_1, \xi_2) &= \boldsymbol{e}^*\, \partial_X \hat{\boldsymbol{r}}(\tilde{X} + \xi_1\boldsymbol{\tau}_1 + \xi_2\boldsymbol{\tau}_2)\,. \end{aligned} \tag{4.14}$$

Practically, the α coefficients of the algebraic bifurcation equation are computed using a basic finite difference scheme (second order, central). A comprehensive derivation of Equation (4.13) can be found in Allgower and Georg [4] and Seydel [17].

Finally, the tangents $\boldsymbol{t}$ can be computed as $\boldsymbol{t} = \xi_1\boldsymbol{\tau}_1 + \xi_2\boldsymbol{\tau}_2$.

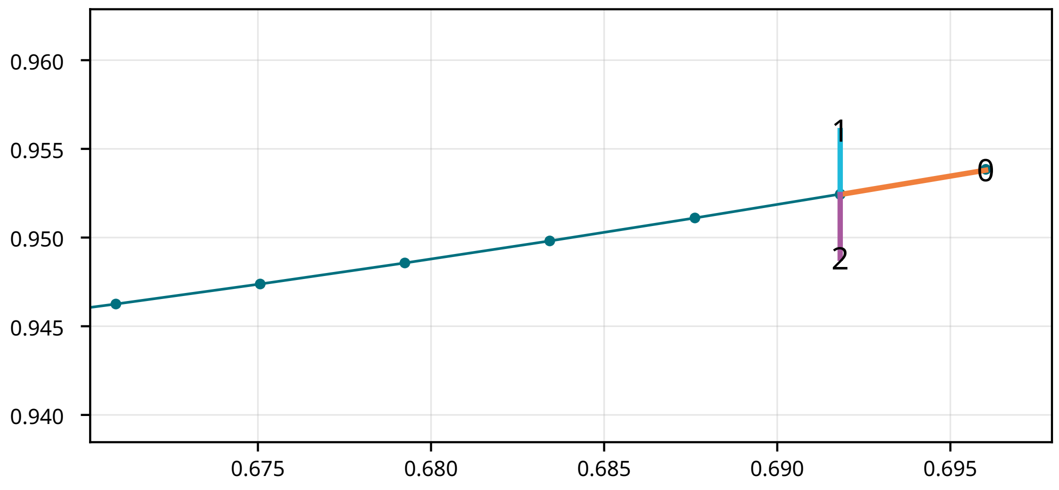
Interactive Branch Selection

The branch tangents method has been implemented to function in both static and interactive modes. In static mode, the entire continuation is performed without any user intervention. During this mode, the tangents are computed at each detected branching point and stored for potential use in post-processing. The continuation is then executed by following the simple jump procedure described in Section 4.2.1.

In interactive mode, the continuation execution is halted at each detected branching point. The current state of the continuation is displayed on a graph, and the user is invited to select one of the possible tangents that appear on the graph. The continuation then resumes in the selected direction.

Figure 4.17 illustrates the application of the branch tangents method on the Lazarus example in interactive mode.

As previously explained in Section 4.3.3, the graph displayed in Figure 4.17 is directly generated from pyHarm during interactive use. Consequently, no quantities or units are assigned to the axes.



```
...
solution converged at om=0.6876220961700652
solution converged at om=0.6918204312361893
solution converged at om=0.6960238287388201
Which branch to follow (integer, look at the graph) ? 0
...
```

Figure 4.17 Example of interactive usage of pyHarm on the Lazarus example. The graph shown here appears during the solving process when the first branching point has been detected.

5 Conclusions

AT THIS STAGE of the thesis, conclusions can be drawn regarding the work that has been accomplished. In this context, the first section of this chapter compiles the achievements and key findings established in Chapter 3 and Chapter 4. Following this, the second section critically discusses the work conducted in this thesis, providing an introspection on the results obtained.

5.1 Summary of the Main Achievements

Chapter 3 — The rotor toy model provided by Safran Tech has been successfully simulated using pyHarm. The initial problem was reproduced, specifically the failure of numerical continuation in the case of asymmetric bearing support. A preliminary solution was proposed, which involved simply increasing the number of sample points used in the AFT procedure.

Chapter 4 — A new nonlinear Newton-Raphson solver has been implemented. This solver performs correction steps based on the pseudo-inverse of the Jacobian matrix of the HB equations. Additionally, a new tangent predictor has been created, with the prediction of the tangent executed through a QR decomposition of the Jacobian matrix.

These newly implemented components facilitated the refactoring of the bifurcation detection and branching mechanism. The new mechanism was tested against two different examples from the literature and demonstrated proper functionality.

A global perturbation method was developed, leveraging the newly created nonlinear solver. This method allows for switching off from the main solution branch and has been shown to enable the discovery of secondary branches of solutions in one example.

The global perturbation method was subsequently refined into a local perturbation method. This approach allows for the application of a perturbation to the HB equations locally, in the vicinity of a detected branching point. To achieve this, the main continuation procedure of pyHarm was modified.

An interactive mode was added to this technique, enabling users to select the perturbation parameter to be applied to the solved system on the fly. As a result, it is now possible to clearly extract a specific secondary branch of solution from the main continuation.

To further enhance the flexibility of pyHarm in branch selection, an additional method was developed that does not rely on introducing small perturbations into the solved system. Instead, the tangents of the secondary branches emanating from a branching point are now computed through the algebraic bifurcation equation.

Finally, this method has been made interactive, allowing users to easily select the branch of solution to be followed by the ongoing numerical continuation.

5.2 Introspection on the Obtained Results

Chapter 3 — The analysis of the initial rotor system could have been further explored. Given that only the global structural matrices of the problem were provided, it was challenging to conduct a more detailed analysis of the system. However, it would have been interesting to attempt to recreate the rotor model from scratch and to vary the structural parameters to observe the physical implications.

Chapter 4 — The branch-switching algorithms presented in Chapter 4 should have been tested against a broader range of examples. So far, the algorithms have only demonstrated functionality on the Duffing oscillator from Lazarus and Thomas [11]. It was initially planned to test these new methods on the Duffing oscillator from Petrov [14] as well. Unfortunately, the results obtained were too unstable. More specifically, the computed continuation in pyHarm faced significant challenges in following the main solution branch. Consequently, the results could not be included in this document.

Similarly, the branch-switching methods could have been applied to the initial rotor toy model from Safran Tech. However, the results in this case were also far from satisfactory.

Overall, the continuation algorithms presented in Chapter 4 have proven effective only on a single Duffing oscillator. This serves as a reasonable starting point for further enhancements in bifurcation handling for pyHarm, but their robustness requires substantial improvement.

On another note, it would have been valuable to study the stability of the system using Floquet theory. This topic was only briefly touched upon in section Section 2.3. However, as noted in that section, bifurcation and stability of dynamical systems are closely related. Implementing a stability study through the computation of the Hill's matrix of the system would have been a significant addition to this thesis.

A
pyHarm Code Organization

*C*ONCRETE IMPLEMENTATION DETAILS related to the pyHarm software have been intentionally kept concise in the main chapters of this thesis to prevent overwhelming the primary discussion with excessive technical information. This appendix provides additional insights into the pyHarm Python code.

The first section outlines the overarching philosophy and design principles guiding the code organization in pyHarm. The second section describes how the contributions of this thesis were integrated into the pyHarm project, with a focus on maintaining consistency and modularity with the existing codebase.

A.1 pyHarm Desing Principles

It has been briefly mentioned in Section 2.2.1 that estimating the Jacobians $\partial_{\hat{\mathbf{q}}}\hat{\boldsymbol{r}}$ constitues the solving part that demands the most computational efforts.

The pyHarm's authors are aware of this cuplprit, and consequently, a substantial part of the codebase is devoted to building the Jacobian of the entire mechanical system under study, and to evaluating them as efficiently as possible.

The philosophy behind the code is to treat the mechanical system as an assembly of elementary elements/connectors such that their contribution to the residual and jacobian can be evaluated independantly. The code is extensively using the factory design pattern in the subpackages to introduce abstract and flexibility when developing new components. The principle of work of a factory pattern is illustrated in Figure A.1.

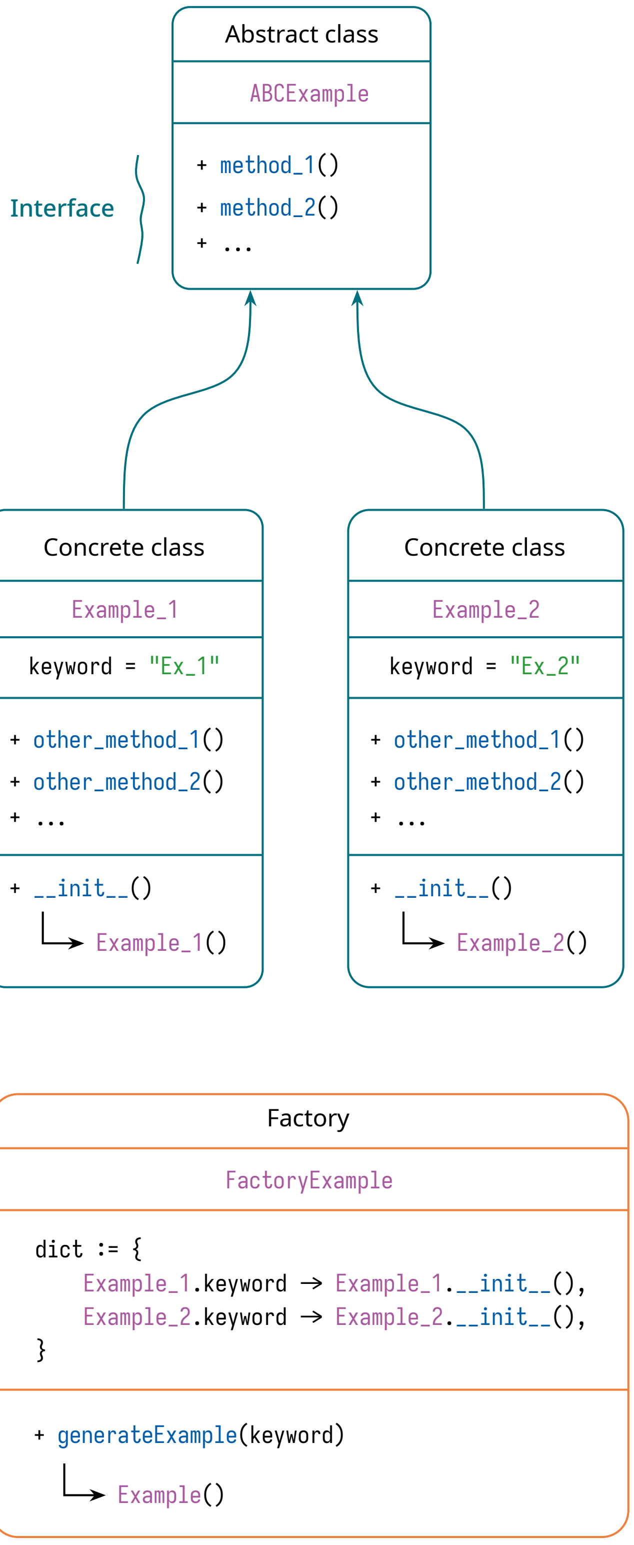


Figure A.1 Use of the factory pattern in pyHarm.

A.2 New Bifurcation Handler Classes

The abstract base class used to define all the bifurcator object is exposed in Figure A.2. The integration of each bifurcator object into the main continuation procedure of pyHarm (`FRF_nonlinear.solve()` function) is described in Figure A.3.

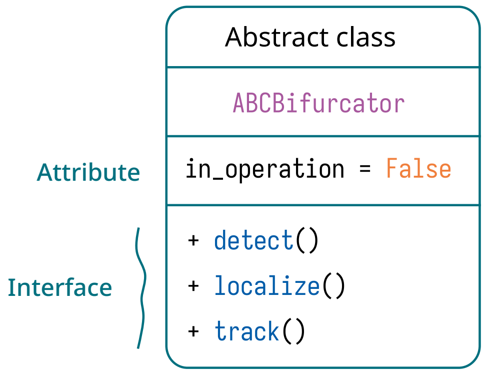


Figure A.2 Bifurcator abstract base class, establishing the common structure that all concrete Bifurcator objects must follow.

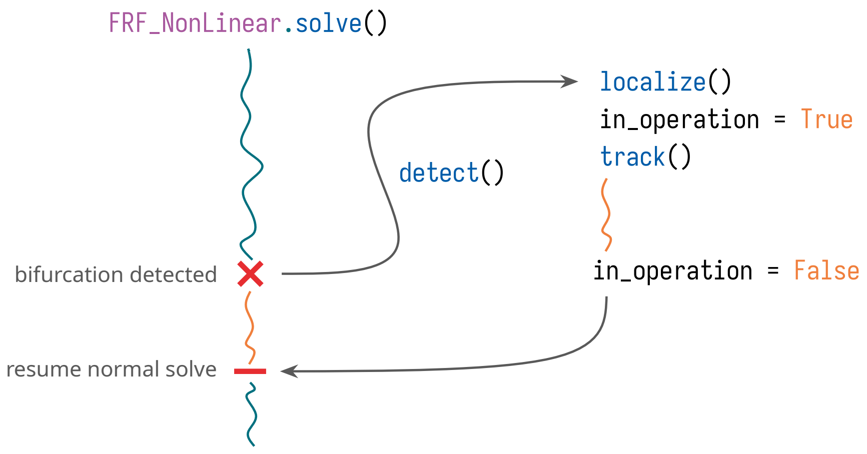


Figure A.3 Integration of the bifurcations handlers into the main analysis.

This page intentionally left blank.

B
Practical Use of pyHarm

From a user perspective, the most effective way to engage with pyHarm is through the completion of the dedicated tutorials. These tutorials are accessible in the same repository as the pyHarm source code. They comprise seven lessons presented as interactive Jupyter notebooks, providing an overview of the capabilities of pyHarm as a nonlinear analysis tool. A strong prior knowledge of nonlinear dynamics is not required; all essential concepts related to the theory behind pyHarm are briefly explained within the notebooks.

This appendix serves as a supplementary resource to these tutorials. Initially, the analysis of a simple Duffing oscillator using pyHarm is presented as an introductory example. All steps leading to the computation of the frequency response of this system are detailed comprehensively.

Subsequently, several utilities that assist in the post-processing of pyHarm results are introduced. The introductory example, along with these post-processing utilities, also aims to clarify how the various figures in this thesis, derived from pyHarm, have been obtained.

Finally, the last section of this appendix provides the explicit implementation of the Safran rotordynamics model in pyHarm, as presented in Section 1.2.1 and analyzed in Chapter 3.

B.1 A Simple Duffing Oscillator

The basic workflow in pyHarm consist in

1. encoding the system under study in a format that is understandable by pyHarm,
2. specifying the type of analyses that should be conducted,
3. runnig all the analyses of the specified problem, and finally
4. post-processing the computed solutions.

In the remainder of this section, the practical use of pyHarm is demonstrated through the example of a simple single-DOF Duffing oscillator. This corresponds to the nonlinear system presented in Section 1.1.2, which is mathematically expressed by Equation (1.1).

B.1.1 Problem Encoding

```python
"""Context: pyharm linear vs duffing"""

```

```python
import numpy as np

from pyHarm.Maestro import Maestro
from pyHarm.DynamicOperator import compute_DFT


# Define the Duffing oscillator
# m*q_ddot + c*q_dot + k*q + gamma*q^3 = f(t)
linsys = dict()
linsys["M"] = np.array([1])
# linsys["C"] = np.array([0.05])  # Lazarus
linsys["C"] = np.array([0.3])  # Tuto 1
linsys["K"] = np.array([1])
linsys["G"] = 0 * linsys["M"]
gamma = 1
f = 1

# Build the global dictionary containing all
# the data of the problem.
DATA = {
    "analysis": {
        "duffing_frf": {
            "study": "frf",
            "puls_inf": 0.01,
            "puls_sup": 3.0,
            "ds0": 5e-3,
            "ds_min": 1e-8,
            "ds_max": 2e-2,

            "sign_ds": 1,
            "verbose": True,
            "stepsizer": "acceptance",
            "predictor": "tangent",
            "corrector": "arc_length",
            "stopper": "bounds",
        },
    },
    "system": {
        "type": "Base",
        "nti": 2048,
        "adim": {
            "status": False,
            "lc": 1.0,
            "wc": 1.0,
        },
    },
    "substructures": {
        "duffing": {
            "matrix": linsys,
            "ndofs": 1,
        },
    },
    "connectors": {
        "loading": {
            "type": "CosinusForcing",
            "connect": {"duffing": [0]},
            "dirs": [0],
            "amp": f,
        },
        "cubic_spring": {
            "type": "CubicSpring",
            "connect": {"duffing": [0]},
            "dirs": [0],
            "k": gamma,
        },
    },
}
```

B.1.2 Solving and Post-Processing

```
def main():

    # Linear case, and duffing with 1 and 12 harmonics.
    nh_list = [1, 1, 12]
    gamma_list = [1E-8, 1, 1]
    label_list = ["linear", r"$n_h=1$", r"$n_h=12$"]

    freqs_list = []
    ampls_list = []
    for nh, gamma in zip(nh_list, gamma_list):
        freqs, ampls = solve_one(nh, gamma)
        freqs_list.append(freqs)
        ampls_list.append(ampls)

    sol = (freqs_list, ampls_list, label_list)

    plot(*sol)

    return sol


def solve_one(nh, gamma):
    # Update problem definition
    DATA["system"]["nh"] = nh
    DATA["connectors"]["cubic_spring"]["k"] = gamma

    # Maestro is the main user interface to pyHarm.
    #
    # It receives the comprehensive problem definition
    # as a global dictionary, then build the system and
    # returns it as a pyHarm Maestro object.
    M = Maestro(DATA)

    # These are the two only methods defined for a Maestro object.
    #
    # getIndex() returns the explicit dofs of the system, for the desired
    # substructure name, node number and direction.
    # These explicit dofs can be consulted with `M.system.expl_dofs`.
    #
    # operate() run all the analysis that were specified in the
    # input dictionary, and stores all the results in the Maestro object.
    # operate() can be put in the shown try-except block.
    # This allows to abort the computations with <CTRL-C> while preserving
    # the already computed values.
    # This can be useful when pyHarm is stuck in a PC infinite loop.
    ix_dof = M.getIndex("duffing", 0, 0)
    try:
        M.operate()
    except KeyboardInterrupt:
        pass

    # Retrieve solutions.
    # `SolList` is the list of all `Solver.SystemSolution` comuputed during
    # the continuation of a prescribed frf analysis (here, "duffing_frf" analysis).
    # Each particular solution is flagged as accepted if the nonlinear solver
    # used for the corrector steps has converged.
    sol_accepted = [sol for sol in M.nls["duffing_frf"].SolList if sol.flag_accepted]
    # Field `x` contains all the balanced Fourier coefficients of the solution.
    # The frequency at wich the solution is computed is appended afterwards.
    freqs = np.array([sol.x[-1] for sol in sol_accepted])  # Extract the Fourier coefficients
```

```python
    n_sol = freqs.shape[0]

    # Discrete Fourier Transform Operator
    # They are available for each system elements,
    # but are actually computed by pyHarm.DynamicOperator.compute_DFT().
    # DFTO = M.nls["duffing_frf"].system.LE_nonlinear_nodlft[0].D
    DFTO = compute_DFT(M.system.nti, M.system.nh)
    # organize the Fourier coefficients (cos/sin) of the solution,
    # each column representing a solution at a particular excitation frequency.
    cs_accepted = np.concatenate([sol.x.reshape(-1, 1) for sol in sol_accepted], axis=1)
    ampls = np.empty(n_sol)
    for ix_sol in range(n_sol):
        # Filter the cs of the desired dof,
        # and get the time domain representation, i.e., the displacement q.
        displ = cs_accepted[ix_dof, ix_sol] @ DFTO["ft"]
        # Find the maximum of the displacement amplitude
        # for each excitation frequencies, a.k.a. the FRF.
        # RMS value or other norm styles could have been used.
        ampls.append(np.max(displ))

    return freqs, ampls


def plot(freqs_list, ampls_list, label_list) -> None:
    import matplotlib.pyplot as plt
    from ..mplrc import load_rcparams, REPORT_TW

    load_rcparams()
    fig, ax = plt.subplots(figsize=(0.6*REPORT_TW, 0.6*REPORT_TW))

    for freqs, ampls, label in zip(freqs_list, ampls_list, label_list):
        ax.plot(freqs, ampls, label=label)

    ax.set_xlabel("Frequency (rad/s)")
    ax.set_ylabel("Amplitude (m)")
    ax.legend()
    fig.show()
```

B.2 Post-Processing Utilities

Post-processing of computed results remains an area for improvement in pyHarm. The software currently does not provide a straightforward method for plotting a NFRC. This section thus provides a set of utilities that can be used to analyze the solutions saved in a `Maestro` object.

Concretely, the code utilized to study the Safran Tech rotor model in Chapter 3 is provided in the following script.

```python
"""solveSafranModel -- Solve the Safran toy model."""

from typing import NamedTuple

import matplotlib.pyplot as plt
import numpy as np

from pyHarm.Bifurcators.ABCBifurcator import BifurcationType
from pyHarm.DynamicOperator import compute_DFT
from pyHarm.Maestro import Maestro
from pyHarm.Solver import SystemSolution, get_last_solution

from .mplrc import load_rcparams, REPORT_TW
from .SafranModel import CHUNCK_LIST, MODEL, SYSTEM

```

```python

class Continuation(NamedTuple):
    M: Maestro  # Maestro object created for the first continuation chunck.
    sol_list: list[SystemSolution]  # Gathered SolList of all the continuation chuncks.
    chunk_list: list[dict]  # "analysis" keys of all the continuation chuncks.


def _get_dof_idx(M: Maestro):
    """Get SystemSolution indexes of the required dofs.

    That is, the DOFs located at the gap bearing.
    DOFs 4 and 5 in pyHarm are DOFs 5 and 6 in the Safran model schematic.
    Nonlinear bearing is located between these two DOFs.
    """
    idx_51 = M.getIndex("linear_rotor", 4, 0)  # shaft node, X-dir
    idx_52 = M.getIndex("linear_rotor", 4, 1)  # shaft node, Y-dir
    idx_61 = M.getIndex("linear_rotor", 5, 0)  # bearing node, X-dir
    idx_62 = M.getIndex("linear_rotor", 5, 1)  # bearing node, Y-dir

    return idx_51, idx_52, idx_61, idx_62


def _filter_harm(idx: np.ndarray[int], ih: list[int]) -> np.ndarray[int]:
    """Filter a SystemSolution index vector, to select only the desired harmonics."""
    expl_ih = np.array([[2 * i - 1, 2 * i] for i in ih], dtype=int).ravel()
    # expl_ih = expl_ih[expl_ih >= 0]  # remove the -1 generated for potential 0-harm

    filtered = np.zeros(idx.shape, dtype=int)
    filtered[expl_ih] = idx[expl_ih]

    return filtered


def _filter_harm_tuple(indexes: tuple, ih: list[int]) -> tuple:
    return tuple((_filter_harm(idx, ih) for idx in indexes))


def compute_amplitude(cont: Continuation, ih=None, pred=False) -> np.ndarray[float]:
    """Compute gap amplitude at the bearing, for specified harmonics."""
    # Discrete fourier transform operator
    DFTO = compute_DFT(cont.M.system.nti, cont.M.system.nh)
    # Get indexes of relevant dofs and associated harmonics.
    indexes = _get_dof_idx(cont.M)

    if ih is None:
        filt_51, filt_52, filt_61, filt_62 = indexes
    else:
        filt_51, filt_52, filt_61, filt_62 = _filter_harm_tuple(indexes, ih)

    def select_pred(sol: SystemSolution):
        return sol.x_pred if pred else sol.x

    return np.array(
        [
            np.linalg.norm(
                (
                    np.sqrt(
                        ((select_pred(sol)[filt_51] - select_pred(sol)[filt_61]) @ DFTO["ft"]) ** 2
                        + ((select_pred(sol)[filt_52] - select_pred(sol)[filt_62]) @ DFTO["ft"]) ** 2
                    )
                )
                @ DFTO["tf"]
            )
            for sol in cont.sol_list
        ]
    )


def extract_accepted(cont: Continuation):
    cont_accepted = Continuation(
        M=cont.M,
        sol_list=[sol for sol in cont.sol_list if sol.flag_accepted],
        chunk_list=cont.chunk_list,
    )
    cont_rejected = Continuation(
        M=cont.M,
        sol_list=[sol for sol in cont.sol_list if not sol.flag_accepted],
        chunk_list=cont.chunk_list,
    )

    return cont_accepted, cont_rejected


def extract_bifurcation(cont: Continuation):
    cont_bifurcation = Continuation(
        M=cont.M,
        sol_list=[sol for sol in cont.sol_list if sol.bifurcation_type is not None],
        chunk_list=cont.chunk_list,
    )
    cont_fold = Continuation(
        M=cont.M,
        sol_list=[
            sol for sol in cont_bifurcation.sol_list if sol.bifurcation_type is
            ↪  BifurcationType.FOLD
        ],
        chunk_list=cont.chunk_list,
    )
    cont_branching = Continuation(
        M=cont.M,
        sol_list=[
            sol for sol in cont_bifurcation.sol_list if sol.bifurcation_type is
            ↪  BifurcationType.BRANCHING
        ],
        chunk_list=cont.chunk_list,
    )
    cont_unknown = Continuation(
        M=cont.M,
        sol_list=[
            sol for sol in cont_bifurcation.sol_list if sol.bifurcation_type is
            ↪  BifurcationType.UNKNOWN
        ],
        chunk_list=cont.chunk_list,
    )

    return cont_bifurcation, cont_fold, cont_branching, cont_unknown


def continuation_plotter():
    fig, ax = plt.subplots(figsize=(REPORT_TW, 0.5*REPORT_TW))
    fold_style = {"color": "C1", "s": 50, "zorder": 2.5, "marker": "x"}
    branching_style = {"color": "C6", "s": 50, "zorder": 2.5, "marker": "x"}
    unknown_style = {"color": "C3", "s": 50, "zorder": 2.5, "marker": "x"}
    ax.scatter([], [], **fold_style, label="fold bifurcation")
    ax.scatter([], [], **branching_style, label="branching point")
    ax.scatter([], [], **unknown_style, label="tangent flip")

    def plot(cont: Continuation, ih=None, pred=False):
        cont_accepted, cont_rejected = extract_accepted(cont)
        cont_bifurcation, cont_fold, cont_branching, cont_unknown =
        ↪  extract_bifurcation(cont_accepted)

        if len(cont_accepted) != 0:
            om_accepted = np.array([sol.x[-1] for sol in cont_accepted.sol_list])
            ampl_accepted = compute_amplitude(cont_accepted, ih)
            if pred:
                pred_style = {"color": "C7", "marker": "."}
                om_pred_accepted = np.array([sol.x_pred[-1] for sol in
                ↪  cont_accepted.sol_list[:-1]])
                ampl_pred_accepted = compute_amplitude(cont_accepted, ih, pred=True)[:-1]
                ax.scatter(om_pred_accepted, ampl_pred_accepted, **pred_style, label="prediction")
                for i in range(len(om_accepted) - 1):
                    ax.plot(
                        [om_pred_accepted[i], om_accepted[i]],
                        [ampl_pred_accepted[i], ampl_accepted[i]],
                        color="C7",
                        linewidth=0.8,
                    )
            if ih is None:
                label = f"nh = {cont.M.system.nh}"
            else:
                label = f"ih = {ih}"
            ax.plot(om_accepted, ampl_accepted, label=label, marker=".")
            # ax.plot(om_accepted, ampl_accepted)

        if len(cont_fold) != 0:
            om_fold = [sol.x[-1] for sol in cont_fold.sol_list]
            ampl_fold = compute_amplitude(cont_fold, ih)
            ax.scatter(om_fold, ampl_fold, **fold_style)

        if len(cont_branching) != 0:
            om_branching = [sol.x[-1] for sol in cont_branching.sol_list]
            ampl_branching = compute_amplitude(cont_branching, ih)
            ax.scatter(om_branching, ampl_branching, **branching_style)

        ax.set_xlabel("Frequency (rad/s)")
        ax.set_ylabel(r"$\|\Delta x\|/g$")
        ax.legend()
        fig.show()

    return plot


def plot_orbit(cont: Continuation, **kwargs) -> None:
    """Orbital motion at the nonlinear bearing, at the desired curve point.

    Keyword args:
        id (int) or om, ampl (float, float):
          Either the index (id) of the solution in the provided list,
          or a point (om, ampl) at which the nearest solution on the curve will be used.
        scale (float):
          Optional scaling factor used to exacerbate the radius values.
    """
    # Discrete fourier transform operator
    DFTO = compute_DFT(cont.M.system.nti, cont.M.system.nh)
    # Get indexes of relevant dofs and associated harmonics.
    idx_51, idx_52, idx_61, idx_62 = _get_dof_idx(cont.M)

    # Check inputs
    if len(cont.sol_list) == 0:
        print("Empty list of solution: unable to plot orbit")
        return
    if not kwargs:
        print("specify either an index `id` or an approximate curve point `om, ampl`")
        return

    # Get id, om, ampl
    om_list = [sol.x[-1] for sol in cont.sol_list]
    ampl_list = compute_amplitude(cont)
    if "id" in kwargs:
        id = kwargs["id"]
    else:
        om_target = kwargs["om"]
        ampl_target = kwargs["ampl"]
        curve_sdist = np.array(  # square distance from the curve
            [
                (om_i - om_target) ** 2 + (ampl_i - ampl_target) ** 2
                for (om_i, ampl_i) in zip(om_list, ampl_list)
            ]
        )
        id = np.argmin(curve_sdist)
    om = om_list[id]
    ampl = ampl_list[id]

    # Polar repr of the orbit at the defined id.
    dx = (cont.sol_list[id].x[idx_51] - cont.sol_list[id].x[idx_61]) @ DFTO["ft"]
    dy = (cont.sol_list[id].x[idx_52] - cont.sol_list[id].x[idx_62]) @ DFTO["ft"]
    r = np.sqrt(dx**2 + dy**2)
    theta = np.arctan2(dy, dx)
    if "scale" not in kwargs:
        scale = 1
    scale = kwargs["scale"]
    mean = np.mean(r)
    r_scaled = mean + (r - mean) * scale

    fig_orbit, ax_orbit = plt.subplots(subplot_kw={"projection": "polar"})
    ax_orbit.plot(
        np.linspace(0, 2 * np.pi, len(r)), 1 * np.ones_like(r),
        color="C7", linewidth=1, linestyle="--", label="Gap",
    )
    ax_orbit.plot(
        np.linspace(0, 2 * np.pi, len(r)), mean * np.ones_like(r),
        color="C2", linewidth=2, label="Mean",
    )
    ax_orbit.plot(
        theta, r_scaled,
        color="C0", linewidth=2, label="Rotor motion",
    )
    ax_orbit.scatter(
        theta[r > 1], r_scaled[r > 1],
        s=15, marker="o", zorder=2.5, color="C6", label="displacement > gap",
    )
    ax_orbit.set_title(f"Freq : {om:.4f} Hz --- Mean ampl. : {ampl:.4f}")
    ax_orbit.set_xticklabels([])
    ax_orbit.set_yticklabels([])
    ax_orbit.spines["polar"].set_visible(False)
    ax_orbit.legend(fontsize=9, loc=10)
    fig_orbit.show()


def solve(nh_list: list[tuple[int, int]], chunck_list: list[dict]) -> list[Continuation]:
    cont_list = []
    for i, nh in enumerate(nh_list):
        x0 = None
        M_chunck_list = []
        hb_params = {"system": SYSTEM | {"nh": nh}}
        for chunck in chunck_list:
            M = Maestro(MODEL | hb_params | chunck)
            try:
                M.operate(x0)
            except KeyboardInterrupt:
                pass
            last_sol = get_last_solution(M.nls["cont"].SolList)
            x0 = last_sol.x[:-1]
            M_chunck_list.append(M)
        cont_list.append(
            Continuation(
                M=Maestro(MODEL | hb_params | chunck_list[0]),
                sol_list=[sol for M in M_chunck_list for sol in M.nls["cont"].SolList],
                chunk_list=chunck_list,
            )
        )
    return cont_list


def main():
    load_rcparams()

    nh_list = [1]
    chunck_list = CHUNCK_LIST

    cont_list = solve(nh_list, chunck_list)

    plot_cont = continuation_plotter()
    for cont in cont_list:
        plot_cont(cont, pred=True)

    return locals()
```

B.3 Initial Safran Rotordynamics Model

The following Python script specifies the construction of the rotordynamic toy model of Safran Tech, so that it can be treated with pyHarm. Note that the structural matrices are given in a `.mat` file, that is name `Jeffcott_asym.mat` in this file.

```python
"""buildSafranModel -- Build the Safran toy model."""

import pathlib

import numpy as np

MODEL_PATH = pathlib.Path(__file__).parent / "res"

MODEL = {
    "analysis": {},
    "system": {},
    "substructures": {
        "linear_rotor": {  # Unforced, linear rotor toy model from Safran
            "filename": str(MODEL_PATH / "Jeffcott_asym.mat"),
            "ndofs": 4,
        },
    },
    "connectors": {
        # Unbalance force can be modelled as m*d*Om^2*cos(Om*t).
        # To model a rotating unbalance, specify sinusoid unbalance
        # loading in the two transverse directions with a dephase of
        # pi/2 between them.
        "unbalance_dir0": {
            "connect": {"linear_rotor": [1]},
            "dirs": [0],
            "type": "GOForcing",
            "amp": 50e-3,
            "ho": 1,  # cos(Om*t) term
            "dto": 2,  # Om^2 term
            "phi": 0,  # Phase lag
        },
        "unbalance_dir1": {
            "connect": {"linear_rotor": [1]},
            "dirs": [1],
            "type": "GOForcing",
            "amp": 50e-3,
            "ho": 1,  # cos(Om*t) term
            "dto": 2,  # Om^2 term
            "phi": np.pi / 2,  # Phase lag
        },
        "bearing_gap": {
            "connect": {"linear_rotor": [4], "INTERNAL": [5]},
            "dirs": [0, 1],
            "type": "PenaltyBilateralGap",
            "g": 0.15e-3,
            "k": 1e8,
        },
    },
}

# The "nh" field need to be completed
# on the corresponding solving file.
SYSTEM = {
    "type": "Base",
    "nti": 1024,  # NOTE: originally 1024, bt 2048 makes it pass the bif
    "adim": {
        "status": True,
        "lc": 0.15e-3,  # Adimensionalized by the gap clearance
        "wc": 1.0,
    },
}

CHUNCK_ALL = {
    "analysis": {
        "cont": {
            "study": "frf",
            "verbose": True,

            "puls_inf": 0,
            "puls_start": 1,
            "puls_sup": 300,
            "ds0": 2e-1,
            "ds_min": 1e-12,
            "ds_max": 1e-1,
            "sign_ds": 1,

            "reductors": [  # NOTE: originally activated
                {
                    "type": "globalHarmonic",
                    "nh_start": np.array([1]),
                    "err_admissible": 1e10,
                    "h_always_kept": np.array([1]),
                    "verbose": False,
                },
            ],

            "stepsizer": "acceptance",  # NOTE: originally acceptance
            "predictor": "tangent",  # NOTE: originally tangent
            "corrector": "arc_length",  # originally arc_length
            "bifurcator": "perturbation",
            "bifurcator_options": {"blind_angle": 180},
            "stopper": "bounds",
            "solver": "NewtonRaphson",
            # "solver": "scipyroot",
            # "solver_options": {"max_iter": 4},
        },
    },
}

# Chained continuations
# CHUNCK_1 -> ... -> CHUNCK_N  => CONT_LIST

CHUNCK_1 = {
    "analysis": {
        "cont": {
            "study": "frf",
            "puls_inf": 1.0,
            "puls_start": 1.0,
            "puls_sup": 72.0,
            "ds0": 1e-1,
            "ds_min": 1e-12,
            "ds_max": 1e-1,
            "sign_ds": 1,
            "verbose": True,
            "stepsizer": "myacceptance",
            "predictor": "mytangent",
            "corrector": "arc_length",
            "stopper": "bounds",
            "solver": "scipyroot",
        },
    },
}

CHUNCK_2 = {
    "analysis": {
        "cont": {
            "study": "frf",
            "puls_inf": 70.0,
            "puls_start": 72.0,
            "puls_sup": 76.0,
            "ds0": 5e-3,
            "ds_min": 1e-12,
            "ds_max": 1e-2,
            "sign_ds": 1,
            "verbose": True,
            "stepsizer": "myacceptance",
            "predictor": "mytangent",
            # "reductors": [
            #     {
            #         "type": "globalHarmonic",
            #         "nh_start": np.array([1]),
            #         "err_admissible": 1e10,
            #         "h_always_kept": np.array([1]),
            #         "verbose": False,
            #     },
            #     {
            #         "type": "AllgowerPreconditioner",
            #     },
            # ],
            "corrector": "arc_length",
            "stopper": "bounds",
            "solver": "scipyroot",
        },
    },
}

CHUNCK_LIST = [CHUNCK_ALL]
```

This page intentionally left blank.

Bibliography

[1] Michel Geradin and Daniel J. Rixen. *Mechanical Vibrations, Theory and Application to Structural Dynamics*. 3rd ed. John Wiley & Sons, Ltd, 2015. ISBN: 978-1-118-90020-8.

[2] Roberto Alcorta et al. “Numerical continuation and stability of nonlinear systems with distributed delays: Application to fluid-induced impacts of tubes in cross-flow”. In: *International Journal of Non-Linear Mechanics* 161 (May 2024), p. 104667. ISSN: 00207462. DOI: 10.1016/j.ijnonlinmec.2024.104667. URL: https://linkinghub.elsevier.com/retrieve/pii/S0020746224000325 (visited on 08/04/2025).

[3] Michael I. Friswell et al. *Dynamics of Rotating Machines*. Cambridge University Press, 2010.

[4] Eugene L. Allgower and Kurt Georg. *Introduction to Numerical Continuation Methods*. Society for Industrial and Applied Mathematics, Jan. 2003. ISBN: 978-0-89871-544-6 978-0-89871-915-4. DOI: 10.1137/1.9780898719154. URL: http://epubs.siam.org/doi/book/10.1137/1.9780898719154 (visited on 08/04/2025).

[5] Jacob A. Dahlke and Robert A. Bettinger. “Practical implementation of pseudo-arclength continuation to ensure consistent path direction”. In: *Acta Astronautica* 215 (Feb. 2024), pp. 205–216. ISSN: 00945765. DOI: 10.1016/j.actaastro.2023.12.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576523006379 (visited on 08/04/2025).

[6] T. Detroux et al. “The harmonic balance method for bifurcation analysis of large-scale nonlinear mechanical systems”. In: *Computer Methods in Applied Mechanics and Engineering* 296 (Nov. 2015), pp. 18–38. ISSN: 00457825. DOI: 10.1016/j.cma.2015.07.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0045782515002297 (visited on 08/04/2025).

[7] Eusebius Doedel. “Numerical analysis of nonlinear equations”. Course Notes.

[8] M. Peeters et al. “Nonlinear normal modes, Part II: Toward a practical computation using numerical continuation techniques”. In: *Mechanical Systems and Signal Processing* 23.1 (Jan. 2009), pp. 195–216. ISSN: 08883270. DOI: 10.1016/j.ymssp.2008.04.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0888327008001027 (visited on 08/04/2025).

[9] Malte Krack and Johann Gross. *Harmonic Balance for Nonlinear Vibration Problems*. Mathematical Engineering. Cham: Springer International Publishing, 2019. ISBN: 978-3-030-14022-9 978-3-030-14023-6. DOI: 10.1007/978-3-030-14023-6. URL: http://link.springer.com/10.1007/978-3-030-14023-6 (visited on 08/04/2025).

[10] Yuri A. Kuznetsov. *Elements of Applied Bifurcation Theory*. Vol. 112. Applied Mathematical Sciences. Cham: Springer International Publishing, 2023. ISBN: 978-3-031-22006-7 978-3-031-22007-4. DOI: 10.1007/978-3-031-22007-4. URL: https://link.springer.com/10.1007/978-3-031-22007-4 (visited on 08/04/2025).

[11] Arnaud Lazarus and Olivier Thomas. "A harmonic-based method for computing the stability of periodic solutions of dynamical systems". In: *Comptes Rendus. Mécanique* 338.9 (Aug. 17, 2010), pp. 510–517. ISSN: 1873-7234. DOI: 10.1016/j.crme.2010.07.020. URL: https://comptes-rendus.academie-sciences.fr/mecanique/articles/10.1016/j.crme.2010.07.020/ (visited on 08/04/2025).

[12] Loïc Peletan et al. "Quasi-periodic harmonic balance method for rubbing self-induced vibrations in rotor–stator dynamics". In: *Nonlinear Dynamics* 78.4 (Dec. 2014), pp. 2501–2515. ISSN: 0924-090X, 1573-269X. DOI: 10.1007/s11071-014-1606-8. URL: http://link.springer.com/10.1007/s11071-014-1606-8 (visited on 08/04/2025).

[13] Loïc Peletan et al. "A comparison of stability computational methods for periodic solution of nonlinear problems with application to rotordynamics". In: *Nonlinear Dynamics* 72.3 (May 2013), pp. 671–682. ISSN: 0924-090X, 1573-269X. DOI: 10.1007/s11071-012-0744-0. URL: http://link.springer.com/10.1007/s11071-012-0744-0 (visited on 08/04/2025).

[14] E. P. Petrov. "Analysis of Bifurcations in Multiharmonic Analysis of Nonlinear Forced Vibrations of Gas Turbine Engine Structures With Friction and Gaps". In: *Journal of Engineering for Gas Turbines and Power* 138.10 (Oct. 1, 2016), p. 102502. ISSN: 0742-4795, 1528-8919. DOI: 10.1115/1.4032906. URL: https://asmedigitalcollection.asme.org/gasturbinespower/article/doi/10.1115/1.4032906/374091/Analysis-of-Bifurcations-in-Multiharmonic-Analysis (visited on 08/04/2025).

[15] E. P. Petrov. "Stability Analysis of Multiharmonic Nonlinear Vibrations for Large Models of Gas Turbine Engine Structures With Friction and Gaps". In: *Journal of Engineering for Gas Turbines and Power* 139.2 (Feb. 1, 2017), p. 022508. ISSN: 0742-4795, 1528-8919. DOI: 10.1115/1.4034353. URL: https://asmedigitalcollection.asme.org/gasturbinespower/article/doi/10.1115/1.4034353/374483/Stability-Analysis-of-Multiharmonic-Nonlinear (visited on 08/04/2025).

[16] Loïc Salles et al. "Dual Time Stepping Algorithms With the High Order Harmonic Balance Method for Contact Interfaces With Fretting-Wear". In: *Journal of Engineering for Gas Turbines and Power* 134.3 (Mar. 1, 2012), p. 032503. ISSN: 0742-4795, 1528-8919. DOI: 10.1115/1.4004236. URL: https://asmedigitalcollection.asme.org/gasturbinespower/article/doi/10.1115/1.4004236/455936/Dual-Time-Stepping-Algorithms-With-the-High-Order (visited on 08/04/2025).

[17] Rüdiger Seydel. *Practical Bifurcation and Stability Analysis*. Vol. 5. Interdisciplinary Applied Mathematics. New York, NY: Springer New York, 2010. ISBN: 978-1-4419-1739-3 978-1-4419-1740-9. DOI: 10.1007/978-1-4419-1740-9. URL: https://link.springer.com/10.1007/978-1-4419-1740-9 (visited on 08/04/2025).

[18] G. Von Groll and D.J. Ewins. "The harmonic balance method with arc-length continuation in rotor/stator contact problems". In: *Journal of Sound and Vibration* 241.2 (Mar. 2001), pp. 223–233. ISSN: 0022460X. DOI: 10.1006/jsvi.2000.3298. URL:

https://linkinghub.elsevier.com/retrieve/pii/S0022460X0093298X (visited on 08/04/2025).

[19] L. Xie et al. "Bifurcation tracking by Harmonic Balance Method for performance tuning of nonlinear dynamical systems". In: *Mechanical Systems and Signal Processing* 88 (May 2017), pp. 445–461. ISSN: 08883270. DOI: 10.1016/j.ymssp.2016.09.037. URL: https://linkinghub.elsevier.com/retrieve/pii/S0888327016303843 (visited on 08/04/2025).

[20] Zipu Yan et al. "Harmonic Balance Methods: A Review and Recent Developments". In: *Computer Modeling in Engineering & Sciences* 137.2 (2023), pp. 1419–1459. ISSN: 1526-1506. DOI: 10.32604/cmes.2023.028198. URL: https://www.techscience.com/CMES/v137n2/53365 (visited on 08/04/2025).

[21] Pau Becerra Zúñiga et al. "Influence of symmetry breaking on the multi-stable responses of a vibro-impact system: Experimental and numerical investigation". In: *Mechanical Systems and Signal Processing* 226 (Mar. 2025), p. 112327. ISSN: 08883270. DOI: 10.1016/j.ymssp.2025.112327. URL: https://linkinghub.elsevier.com/retrieve/pii/S0888327025000287 (visited on 08/04/2025).

[22] Weitao Chen et al. "Stability and bifurcation analysis of a bevel gear system supported by finite-length squeeze film dampers". In: *Nonlinear Dynamics* 100.4 (June 2020), pp. 3321–3345. ISSN: 0924-090X, 1573-269X. DOI: 10.1007/s11071-020-05723-2. URL: https://link.springer.com/10.1007/s11071-020-05723-2 (visited on 08/04/2025).

[23] Rolls Royce. *The Jet Engine*. 5th Edition. Wiley.

[24] E.V. Appleton and Balth. Van Der Pol. "XVI. *On a type of oscillation-hysteresis in a simple triode generator*". In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 43.253 (Jan. 1922), pp. 177–193. ISSN: 1941-5982, 1941-5990. DOI: 10.1080/14786442208633861. URL: https://www.tandfonline.com/doi/full/10.1080/14786442208633861 (visited on 08/05/2025).

[25] Minoru Urabe. "Galerkin's procedure for nonlinear periodic systems". In: *Archive for Rational Mechanics and Analysis* 20.2 (Jan. 1, 1965), pp. 120–152. ISSN: 1432-0673. DOI: 10.1007/BF00284614. URL: https://doi.org/10.1007/BF00284614 (visited on 08/05/2025).

[26] K.S. Kundert and A. Sangiovanni-Vincentelli. "Simulation of Nonlinear Circuits in the Frequency Domain". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 5.4 (Oct. 1986), pp. 521–535. ISSN: 0278-0070. DOI: 10.1109/TCAD.1986.1270223. URL: http://ieeexplore.ieee.org/document/1270223/ (visited on 08/05/2025).

[27] Kenneth C. Hall, Jeffrey P. Thomas, and W. S. Clark. "Computation of Unsteady Nonlinear Flows in Cascades Using a Harmonic Balance Technique". In: *AIAA Journal* 40.5 (May 2002). Publisher: American Institute of Aeronautics and Astronautics, pp. 879–886. ISSN: 0001-1452. DOI: 10.2514/2.1754. URL: https://arc.aiaa.org/doi/10.2514/2.1754 (visited on 08/05/2025).

[28] Bertrand Madet and Quentin Mercier. *pyHarm*. Version 1.1.3. URL: https://gitlab.com/drti/pyharm.

[29] A. Dhooge et al. “New features of the software MatCont for bifurcation analysis of dynamical systems”. In: *MCMDS*. 2nd ser. 14 (2008), pp. 147–175.

[30] T. M. Cameron and J. H. Griffin. “An Alternating Frequency/Time Domain Method for Calculating the Steady-State Response of Nonlinear Dynamic Systems”. In: *Journal of Applied Mechanics* 56.1 (Mar. 1, 1989), pp. 149–154. ISSN: 0021-8936, 1528-9036. DOI: 10.1115/1.3176036. URL: https://asmedigitalcollection.asme.org/appliedmechanics/article/56/1/149/389465/An-Alternating-FrequencyTime-Domain-Method-for (visited on 08/17/2025).

[31] E.H. Moussi. “Computation of nonlinear normal modes of structures with elastic stops”. In: *Proceedings of the 4th Canadian Conference on Nonlinear Solid Mechanics* 23 (July 2013).

[32] Grégoire Bourdouch. *Machine learning for experimental bifurcation analysis*. Master’s thesis. Liège: University of Liège, 2025. URL: http://hdl.handle.net/2268.2/23382.