POLITECNICO DI TORINO

MASTER'S IN AUTOMOTIVE ENGINEERING
DEPARTMENT OF MECHANICAL AND AEROSPACE ENGINEERING

Master's Degree Thesis

Automated Engine Component Design: A Parametric and Computational Framework for the Crankshaft

Supervisor:
Prof. Carlo Rosso

Candidate:
Akshay Suresh
270380

October – 2025

ABSTRACT

This thesis presents a novel computational framework that automates the design and validation of high-performance mechanical components, using the internal combustion engine crankshaft as a comprehensive case study. Addressing the limitations of traditional, manual design processes, this work introduces a unified methodology that seamlessly integrates multi-disciplinary engineering analyses into an iterative design process. The framework systematically conducts kinematic, dynamic, and modal analyses, providing real-time feedback to the user before automatically generating a foundational Computer-Aided Design (CAD) model. A key contribution is the system's ability to perform forced vibration analysis and compute a Campbell Diagram to ensure operational stability and avoid destructive resonance. By significantly reducing the design cycle from weeks to hours and enabling rapid, data-driven design iterations, this work establishes a new paradigm for intelligent design, empowering engineers to explore a greater range of robust design alternatives.

ACKNOWLEDGEMENTS

The successful completion of this thesis is a reflection of the incredible support and encouragement I have received, and I am delighted to have this opportunity to thank the people who made it possible.

My most profound gratitude is for my supervisor, **Professor Carlo Rosso**. His mentorship was fundamental. I am deeply thankful for his sharp insights, which always pushed me to strengthen my arguments, and for his consistent guidance throughout the research process. Thank you for believing in my potential and for your invaluable encouragement.

A special and heartfelt thank you goes to my sister, **Dr. Sneha Suresh**. Her example as a scholar has always been an inspiration, but her solidarity as my sister has been my greatest strength. Thank you for the invaluable discussions that always brought perspective, for pushing me to be my best, and for your constant support, which means the world to me.

I am eternally grateful to my dear **parents**. This achievement is a testament to your endless love and the sacrifices you have made for my education. Thank you for instilling in me the confidence to pursue my dreams and for your boundless faith in me, which often exceeded my own.

I am also grateful for the support of my friends. Their valuable perspective and unwavering encouragement were essential throughout this journey, and their companionship greatly enriched my time here in Turin.

Finally, I wish to thank **Politecnico di Torino** for providing the opportunity and resources to complete this master's degree.

Contents

List of Figures

List of Tables

1. INTRODUCTION

In modern mechanical engineering, the pursuit of optimal design for high-performance components like the internal combustion engine's crankshaft is an endeavour fraught with complexity. Traditional design workflows—often fragmented, manual, and sequential—are no longer sufficient. They force engineers into a labour-intensive process of disconnected analyses, which not only introduces risk of error but also fundamentally restricts innovation. This inability to rapidly iterate and explore a broad design space is a primary bottleneck, hindering the development of lighter, stronger, and more efficient components that are critical for advancing engine performance.

This thesis directly confronts these limitations by pioneering a unified computational framework for the automated design and analysis of crankshafts. This work introduces a new paradigm, integrating the traditionally separate stages of kinematic analysis, structural verification, and dynamic validation into a single, cohesive system managed through an interactive MATLAB GUI. The framework's core contribution is a robust methodology that automates the entire workflow, from initial parameter definition to the automated generation of a foundational CAD model. By embedding sophisticated analytical tools directly into the design loop, this system empowers engineers to make informed decisions rapidly, ensuring the structural and dynamic integrity of the proposed design baseline.

The investigation begins by establishing the theoretical foundation, reviewing the principles of crankshaft dynamics, stress analysis, and torsional vibrations. Subsequently, the architecture of the proposed computational framework is detailed, from its user-centric interface to its underlying algorithms. The framework's practical capabilities are then demonstrated through a comprehensive case study, validating its ability to generate an optimized crankshaft design based on a set of operational requirements. The thesis concludes with an analysis of the results, a discussion of the work's broader implications for component design, and an outline of promising directions for future work.

2. Literature Review

The design of an internal combustion engine's crankshaft is a highly complex, multi-disciplinary challenge. Historically, this process was characterised by a fragmented and time-intensive workflow, where separate engineering teams would sequentially perform uncoupled analyses for kinematics, structural integrity, and dynamics [1]. Such a methodology created significant risk, as critical design flaws, particularly those related to dynamic behaviour like torsional vibration, were often discovered only late in the development cycle, leading to costly delays and redesigns [2, 3]. This traditional paradigm, reliant on disconnected software tools and manual data transfer, proved insufficient for the demands of modern engine development, which requires rapid iteration and optimization [1, 4].

In response to these limitations, a new design philosophy has emerged, centred on the development of integrated computational frameworks that automate and unify the entire design workflow [4, 5]. The primary objective of this modern approach is to condense a design cycle that once took weeks into a matter of hours by establishing a cohesive, data-centric process [1]. By integrating the traditionally separate stages of analysis and design, these frameworks ensure the structural and dynamic integrity of the component is considered from the earliest conceptual stages, bridging the gap between theoretical calculation and physical design [4]. This review will examine the key pillars that constitute such an integrated framework, as identified in the literature.

2.1. Foundational Analysis and Parametric Sizing

The foundation of a modern, automated design process is a robust analytical core that performs the initial sizing of the component based on high-level engine parameters [5]. This "first attempt" design phase begins with a rigorous kinematic and dynamic load analysis to accurately model the forces acting on the mechanism, including inertia forces from reciprocating masses and gas forces derived from the in-cylinder combustion pressure [5, 7]. The output of this initial stage is a set of fundamental geometric dimensions for the crankshaft, such as journal diameters, web thicknesses, and crankpin lengths, which are calculated using well-established analytical and empirical formulas from engineering literature [5, 8]. This data-driven approach

provides a structurally sound baseline for the component, which can then be passed to subsequent stages for detailed verification and refinement [1, 4].

2.2. Advanced Verification through Integrated CAE

While an initial sizing provides a starting point, it is insufficient to guarantee the component's performance under complex operating conditions. The literature strongly indicates that an effective framework must integrate advanced Computer-Aided Engineering (CAE) for detailed verification, focusing primarily on dynamic behaviour and fatigue life [4].

Torsional vibration remains a primary cause of crankshaft failure in the field [2]. Real-world operating conditions, such as engine misfire or compressor valve failures, can introduce severe, "non-ideal" torque harmonics that are not predicted by simplistic models [2]. Therefore, a critical function of an integrated framework is to perform a modal analysis on the generated geometry to identify the crankshaft's natural frequencies. These frequencies can then be plotted on a Campbell Diagram to check for dangerous intersections with engine excitation orders, a process demonstrated by Rosso et al. [4]. The ability to perform this check automatically and then feedback any necessary design modifications—such as altering pin diameters to shift a resonant frequency—is a core advantage of a closed-loop design system [4].

Fatigue is the other critical failure mode, with cracks often originating at stress concentration points, such as the fillets between the journals and webs [9, 10]. The fatigue strength of these fillets is commonly enhanced by manufacturing processes like fillet rolling, which induces beneficial compressive residual stresses [9]. However, this complicates the analysis, as research by Spiteri et al. has shown that the compressive stresses can cause small surface cracks to arrest, meaning that a simple "surface crack" failure criterion can be overly conservative and may not correlate to a complete, two-piece failure [9]. Furthermore, case studies of in-field failures have shown that even microscopic cracks, such as those induced by an improper grinding process during a repair, can act as stress risers and lead to the rapid destruction of journal bearings and subsequent crankshaft failure [10]. This highlights the need for a design framework to be founded on a deep understanding of the component’s complex stress states and failure mechanisms.

2.3. The Necessity of High-Fidelity 3D Modelling

A recurring theme in the literature is that accurate prediction of a crankshaft's complex dynamic behaviour requires high-fidelity models. Early analytical methods often relied on simplified representations, such as beam element models. However, research by Kang et al. demonstrated through experimental modal analysis that such models are inadequate for capturing the coupled torsional-flexural and longitudinal-flexural vibration modes that are characteristic of a crankshaft's complex geometry [11]. Their work concluded that only detailed solid element Finite Element Models (FEM) show strong correlation with experimental data, making them essential for accurate dynamic analysis [11]. This finding underscores the importance of the final stage of an integrated framework: the automated generation of a complete 3D CAD model. This model serves not only as a visual blueprint but also as the necessary input for the high-fidelity CAE verification required to ensure the design's integrity [1, 4].

2.4. Conclusion of Literature Review

The literature confirms a clear and necessary progression from fragmented, manual design methods to integrated, automated computational frameworks. The most effective of these systems, such as those developed by Delprete, Rosso, and their colleagues, create a seamless workflow from conceptual analysis to detailed design. They achieve this by linking a powerful analytical front-end, often developed in a platform like MATLAB, with a parametric CAD back-end, such as SolidWorks. This linkage is typically achieved through the automated generation of parameter files that can be read by a custom macro, which then programmatically constructs the 3D model [1, 4]. The resulting CAD model enables the use of high-fidelity CAE tools to verify the design against complex failure modes like torsional resonance and fatigue. This closed-loop, iterative process represents the current state-of-the-art and identifies a clear gap for a tool that not only automates these analyses but also makes them accessible through an interactive, user-centric interface. Therefore, this thesis proposes the development of such a framework, the methodology of which will be detailed in the following chapter.

3. Methodology

This chapter outlines a detailed methodology for the analysis, design, and verification of a multi-cylinder crankshaft. The framework is a comprehensive, user-driven system that integrates various engineering disciplines, including solid mechanics, dynamics, and materials science. The process systematically transitions from a simplified physical model to a rigorous dynamic analysis to ensure both the structural integrity and operational stability of the final design.

The entire process is managed through a central user interface (UI), which serves as the primary gateway for all user interaction and provides a clear, traceable workflow. This system is designed with a modular architecture, where each analytical step is encapsulated within a dedicated module. Data is exchanged between these modules via a persistent data layer, ensuring consistency and continuity throughout the analysis. The user navigates through a series of dedicated interfaces to define input parameters, execute computations, and visualize the results.

The central application guides the user through the following key stages:

- Parameter Input and Engine Cycle Definition
- Kinematic and Load Analysis
- Preliminary Dimensioning and Structural Verification
- Dynamic and Vibrational Analysis

3.1. Parameter Input and Engine Cycle Definition

The process begins with the user defining the engine's fundamental characteristics through a dedicated input GUI. This includes geometric parameters such as the crank radius (r) and connecting rod length (l), as well as operating conditions like engine speed (ω) and the number of cylinders (N).

A critical step is the definition of the in-cylinder pressure curve, which is the primary source of dynamic loading on the crankshaft. The framework supports two methods for this:

- **Theoretical Cycles:** For preliminary design, the user can select either an Otto or Diesel cycle. The pressure-volume (P−V) diagram is calculated based on fundamental thermodynamic principles. The resulting pressure-crank angle curve is then saved for subsequent analyses.
- **Experimental Data:** For advanced analysis and validation, the user can import real-world pressure-crank angle data from an external file. This allows for a more accurate representation of the engine's actual operating conditions.

This stage is crucial as all subsequent calculations are dependent on the data established here. A graphical output allows the user to immediately visualize the defined pressure curve, as shown in the dedicated GUI.

3.2. Kinematic and Dynamic Load Analysis

This section details the calculation of the forces and moments acting on the crankshaft. These loads are a combination of inertia forces from the reciprocating components and gas forces from combustion.

3.2.1. Kinematic Equations: The analysis begins with the kinematics of the slider-crank mechanism. The instantaneous position of the piston (x) from Top Dead Centre (TDC) is a function of the crank angle (θ), crank radius (r), and connecting rod length (l) [12]:

$$x = r(1 - \cos\theta) + l\left(1 - \sqrt{1 - \left(\frac{\mathrm{r}}{\mathrm{l}}\right)^2 \sin^2\theta}\right)$$

To simplify the subsequent derivation of forces, a common and highly accurate approximation is used [12]:

$$x \approx r(1 - \cos\theta) + \frac{r^2}{2l}\sin^2\theta$$

The piston's velocity (v) and acceleration (a) are found by taking the first and second, time derivatives of its position [12], where the time derivative of the crank angle is the engine speed $\left(\frac{\mathrm{dt}}{\mathrm{d}\theta} = \omega\right)$:

$$v = \omega r \left(\sin\theta + \frac{r}{2l}\sin(2\theta)\right)$$

$$a = \omega^2 r \left(\cos\theta + \frac{r}{l}\cos(2\theta)\right)$$

This acceleration equation is fundamental, as it directly determines the inertia forces acting on the crankshaft.

3.2.2. Inertia Forces: The reciprocating inertia force (F_{in}) is a significant source of loading, particularly at high engine speeds. It is calculated from the piston acceleration (a) and the mass of the reciprocating assembly (m_{rec}) [12], as follows:

$$F_{in} = m_{rec}\omega^2 r \left(\cos\theta + \frac{r}{l}\cos(2\theta)\right)$$

The first term, proportional to $\cos\theta$, is known as the primary inertia force, while the second term, proportional to $\cos(2\theta)$, is the secondary inertia force. This distinction is critical for engine balancing.

3.2.3. Gas Forces and Torque: The instantaneous gas force (F_g) is derived directly from the instantaneous in-cylinder pressure $(p(\theta))$ and the piston's cross-sectional area (A_p) [12].

$$F_g = p(\theta) \cdot A_p$$

The total force on the piston pin is the sum of the gas force and the inertia force. This force is then decomposed into tangential (F_t) and radial (F_r) components acting on the crankpin [12].

$$F_t = \frac{F_{piston}}{\cos\beta}\sin(\theta + \beta)$$

$$F_r = \frac{F_{piston}}{\cos\beta}\cos(\theta + \beta)$$

The tangential force is used to calculate the single-cylinder engine torque ($M_t = F_t . r$).

3.2.4. Multi-Cylinder Analysis: For multi-cylinder configurations, the framework sums the individual cylinder torques according to the defined firing order and phase angles. This produces a vector of the total engine torque over a full 720∘ cycle. The resultant primary and secondary forces and moments are also calculated to assess the engine's balancing characteristics.

The application GUI provides a graphical representation of the individual and combined moments, highlighting the "most stressed" cylinder and the location of maximum forces. The system generates polar diagrams of the resultant forces and moments. These diagrams plot the magnitude and direction of the force and moment vectors over the 720∘ cycle. They are used to visually inspect the degree of unbalance in the system and to determine the required balancing masses. The first order and second-order resultant forces and moments are analysed separately, as they represent the primary and secondary sources of unbalance, respectively.

The instantaneous engine torque (M_t) for each cylinder is calculated from the tangential component of the combustion and inertial forces. The total engine torque is the sum of the instantaneous torques from all cylinders, considering the firing order [12]:

$$M_{t,total} = \sum_{i=1}^{N} M_{t,i}$$

The mean engine torque ($M_{t,avg}$) is a crucial parameter for determining the engine's power output and for dimensioning the flywheel. It is calculated by integrating the total engine torque over a full cycle.

Specifically, the mean torque is the integral of the total torque over a full four-stroke cycle (720° or 4π radians), divided by the period of the cycle [12]:

$$M_{t,avg} = \frac{1}{4\pi} \int_0^{4\pi} M_{t,total}(\theta)\ d$$

3.3. Preliminary Dimensioning and Structural Verification

This stage of the methodology is dedicated to translating the calculated loads into a physical design and verifying its structural integrity. This module is responsible for calculating the geometric dimensions of the crankshaft and ensuring it can withstand the applied forces and moments. The process is an iterative, user-driven approach that involves an initial dimensioning phase based on empirical formulas, followed by a detailed stress analysis to ensure the design meets both strength and fatigue criteria. This ensures the design is not only computationally sound but also aligned with practical engineering experience.

3.3.1. Dimensioning: Based on the calculated maximum pressure (p_{max}) and empirical formulas from engineering literature, the framework determines an initial set of geometric dimensions. These include the journal diameter (d), lengths (L_1, L_2), and web dimensions (B, H). These preliminary dimensions are based on design rules and ratios relative to the engine's main parameters, such as bore and stroke, to ensure a structurally sound starting point.

The crankshaft's major dimensions—including the crankpin diameter (d), crank web thickness (B), and crank web height (H)—are initially determined as a function of the engine's cylinder bore diameter (D). For example, the crankpin diameter is often set as a fraction of the bore diameter [13]:

$$d = 0.6 \cdot D$$

These initial dimensions serve as a starting point for the design and are subject to verification in the subsequent steps.

3.3.2. Verification of Stresses: A critical stress verification process is performed to calculate the combined stresses at the most susceptible points of failure. The framework evaluates both bending and torsional stresses at two critical crank positions: the point of maximum tangential force and the point of maximum radial force. The calculated stress values are then compared against the material's allowable stress limit (σ_{adm}).

The formula for calculating the combined stress (σ_{comb}) is derived from classical mechanics, considering the bending moment (Mb) and torsional moment (M_t) at each section. The combined stress is given by [14]:

$$\sigma_{comb} = \sqrt{\sigma_b^2 + 4\tau^2}$$

Here, σ_b is the bending stress, and τ is the shear stress. The application presents these results in a verification graphical user interface (GUI), indicating if the design is "OK" or "NOK" and calculating the minimum required dimensions to meet the safety criteria. A dedicated module allows the user to input and verify their own dimensions, with the system providing real-time feedback by calculating the combined stress for the user-defined geometry. This interactive loop is a core innovation of the framework, as it enables users to quickly iterate on the design until it meets all safety requirements.

Beyond static stress, the primary mode of failure for crankshafts is fatigue. The system performs a fatigue analysis by considering the alternating stresses and the material's endurance limit. The minimum required diameters for the crankpin and journal bearings are calculated based on the fatigue stress concentration factors and the allowable alternating stress. The calculation uses the following formula to find the minimum diameter (d_{min}) [14]:

$$d_{min} = \left(\frac{32}{\pi \sigma_{amm}} \sqrt{\left(M_{b,1}\right)^2 + \left(M_{b,2}\right)^2 + 4(M_t)^2} \right)^{1/3}$$

Here, $M_{b,1}$ and $M_{b,2}$ are the maximum bending moments in two perpendicular planes, and Mt is the maximum torsional moment. The designed dimensions are then compared against these minimum required dimensions to ensure the crankshaft has an adequate fatigue life.

3.4. Dynamic and Vibrational Analysis

This stage of the methodology performs a detailed dynamic analysis to ensure the crankshaft's behaviour is safe and reliable across the engine's full operating range. The torsional dynamic analysis is crucial for ensuring the crankshaft does not experience destructive resonant vibrations. This module consists of three main sub-sections: natural frequency analysis, harmonic analysis, and forced vibration response analysis.

3.4.1. Torsional Stiffness and Inertia: The crankshaft is modelled as a multi-degree-of-freedom (MDOF) lumped-mass system. The torsional stiffness (K_{eq}) of each shaft section is calculated using a ponderal mean of multiple established empirical formulas. The equivalent moment of inertia (J_{eq}) for each mass (e.g., flywheel, crankpins, journals) is determined. These values are used to assemble the global mass matrix (M) and stiffness matrix (K) for the entire system.

3.4.2. Modal Analysis: The crankshaft-flywheel system is modelled as a multi-mass torsional system, where the system's dynamics are governed by its moments of inertia and stiffnesses. The natural frequencies and mode shapes are determined by solving the system's equation of motion, which can be expressed as [15]:

$$[J]\{\ddot{\theta}\} + [K]\{\theta\} = \{0\}$$

Here, [J] is the diagonal matrix of mass moments of inertia, and [K] is the stiffness matrix of the shaft sections. The undamped natural frequencies (ω_n) and their corresponding mode shapes (ϕ) are then computed by solving the generalized eigenvalue problem [15]:

$$[K]\phi = \omega_n^2[J]$$

The eigenvalues (ω_n^2) of the system correspond to the natural frequencies, and the eigenvectors (ϕ) correspond to the mode shapes, which define the relative angular displacements of each mass. The analysis also filters out the rigid-body mode, which has a natural frequency close to zero.

To determine the potential for resonance, a Fourier analysis is performed to decompose the total engine torque into its harmonic components. The total engine torque signal is a periodic function of the crank angle. The Fourier series expansion provides the amplitudes (A_k) and phases (ϕ_k) of the different harmonic orders (k) [15]:

$$M_t(\theta) = \sum_{k=1}^{n} A_k \sin(k\theta + \phi_k)$$

The system calculates the harmonic amplitudes and phases for relevant orders (e.g., from order 0.5 up to 6) through numerical integration of the torque signal over a full 720∘ cycle. These harmonics represent the exciting torques that can cause forced vibration.

The results of the dynamic analysis are presented in a Campbell Diagram, which visually identifies potential resonance points where the natural frequencies intersect with the engine's harmonic excitation orders. A dedicated graphical user interface (GUI) displays this diagram, highlighting critical speeds that must be avoided.

The Campbell diagram is a powerful tool for identifying engine speeds at which resonance can occur. It is constructed by plotting the system's natural frequencies (as horizontal lines) against the engine speed (RPM) and superimposing the harmonic excitation lines (lines with slopes proportional to the harmonic order). The points where these lines intersect represent the critical speeds.

The critical speed (ω_{crit}) is determined by the following formula [15]:

$$\omega_{crit} = \frac{\omega_{n,i}}{k}$$

Here, ω_{crit} is the critical speed (in rad/s), $\omega_{n,i}$ is the i-th natural frequency, and k is the harmonic order. At these speeds, the engine's excitation frequency matches a natural frequency of the system, leading to resonance and potentially destructive vibrations. The system automatically plots this diagram, clearly marking the intersection points to provide a critical visual assessment.

3.4.3. Forced Vibration Analysis: To simulate real-world conditions, a forced vibration analysis is conducted. The total engine torque is first decomposed into its harmonic components using a Fast Fourier Transform (FFT), which provides the amplitude and phase of each excitation order. The steady state forced response is then calculated by summing the contributions of each harmonic across all modes, incorporating a representative modal damping factor of $\zeta = 0.02$. This value is a common assumption for the internal hysteretic damping of steel crankshafts when specific experimental data is not available [16]. This analysis provides a plot of the vibration amplitude at the flywheel as a function of engine speed, as well as the dynamic torque in each shaft section, allowing for a complete assessment of the system's dynamic performance.

The system computes the forced vibration response of the crankshaft over a range of engine speeds, determining the amplitude of torsional vibration at each lumped mass due to the harmonic excitation. The response at a specific mass, such as the flywheel or pulley, is calculated by summing the contributions of all harmonic orders and all mode shapes. The amplitude is heavily influenced by the Dynamic Magnification Factor (DMF), which peaks sharply at resonance.

The DMF is given by the following formula [15]:

$$DMF = \frac{1}{\sqrt{\left(1 - \left(\frac{\omega_{excitation}}{\omega_n}\right)^2\right)^2 + \left(2\zeta \frac{\omega_{excitation}}{\omega_n}\right)^2}}$$

Here, ζ is the damping ratio, $\omega_{excitation}$ is the excitation frequency, and ω_n is the natural frequency. The system then plots the vibration amplitude versus engine speed, which allows for a clear identification of speeds where the system's response is unacceptable. This detailed methodology, from defining inputs to analysing dynamic behaviour, provides a robust and thorough framework for the analysis of crankshafts.

3.5. Automated CAD Model Generation

The final stage of the methodology is the automated generation of the crankshaft's 3D CAD model. This process translates the verified geometric and operational parameters from the analytical stages into a tangible solid model. The framework employs a decoupled, file-based automation workflow that leverages the distinct strengths of both MATLAB for computation and SolidWorks for geometric construction. This approach ensures a high degree of reliability and provides a clear separation between the engineering calculations and the CAD execution.

The generation process is divided into two primary phases: data export from MATLAB and automated construction within SolidWorks.

3.5.1. Data Export from MATLAB

Upon completion of the analytical stages within the MATLAB GUI, the user initiates the "Generate CAD" command. This triggers a script that performs two critical functions:

- Generation of a Parametric Equations File: A text file is generated containing all the final, verified geometric dimensions (e.g., journal diameters, lengths, cheek thickness). This file is formatted to be read directly by the SolidWorks Equation Manager, making the model's geometry parametrically controlled by the MATLAB output. This method ensures that any changes in the analysis are automatically propagated to the CAD model upon regeneration.

- Generation of a Procedural Configuration File: A second text file is generated to hold the logical parameters required by the automation macro, such as the number of cylinders and their respective crank throw angles. This decouples the geometric data from the procedural instructions, making the automation script more modular and easier to maintain.

After these files are written, the MATLAB environment launches the SolidWorks application and presents the user with a dialog box containing instructions for the final steps.

3.5.2. Automation within SolidWorks

To ensure maximum reliability and a user-friendly experience, a custom macro is integrated into the SolidWorks toolbar via a dedicated button. Once the user clicks this "Generate Crankshaft" button, the following automated sequence occurs:

- Template Ingestion: The macro opens a universal crankshaft template part. This template is pre-linked to the external equations file, causing it to immediately update its base features to match the verified dimensions from MATLAB.

- Procedural Generation: The macro reads the procedural configuration file to determine the number of cylinders and their phase angles.

- Geometric Replication Loop: The macro programmatically executes a loop based on the number of cylinders. In each iteration, it procedurally replicates and positions the constituent geometric bodies—the crank throws and journals—into their correct positions and orientations along the primary axis of the crankshaft.

- Boolean Union Operation: Following the geometric replication loop, the macro executes a final Boolean union operation. All constituent solid bodies are programmatically identified and aggregated before being merged into a single, monolithic solid part, completing the automated generation of the crankshaft's core geometry.

3.5.3. Final Manual Operations

As a final step, the user is prompted to apply fillets to the journal and crankpin edges. The required fillet radii, having been defined in the MATLAB GUI and exported to the SolidWorks Global Variables via the equations file, are readily available within the Fillet feature manager. This streamlines the final manual task, ensuring design consistency and accuracy while providing the user with an opportunity for a final visual inspection of the model.

4. Results and Discussion

This chapter presents the practical application and validation of the computational framework detailed in the preceding methodology. The primary objective is to demonstrate the framework's end-to-end capability to translate a set of high-level engineering requirements into a fully analysed and geometrically defined crankshaft.

To achieve this, a comprehensive case study was conducted on a standard 2.0-liter, four-cylinder, four-stroke gasoline engine. This common configuration serves as a representative test case to verify the robustness and accuracy of each analytical module within the integrated system.

The results are presented sequentially, mirroring the user's workflow through the MATLAB graphical user interface (GUI). The chapter begins by defining the initial input parameters for the case study engine. It then presents the outputs from the kinematic and dynamic load analysis, followed by the results of the structural and fatigue verification.

Subsequently, the findings from the torsional vibration analysis are detailed, including the identification of critical speeds. The chapter culminates with the final output of the framework: the automatically generated 3D CAD model of the crankshaft. Each step is supported by figures generated directly by the framework, providing a clear and objective validation of its functionality.

4.1. The User Interface and Case Study Definition

The entire analysis is managed through a central graphical user interface (GUI), which serves as the primary portal for all user interactions. As shown in Figure 4.1, the main screen presents the user with a clear, modular workflow, organized by the sequential buttons on the left-hand side. This design guides the user through each major stage of the analysis, from initial 'Input' to final 'Generate CAD'.

The validation process begins by engaging the first module, 'Input', to define the engine's fundamental parameters. For this case study, a conventional inline four-cylinder Spark Ignition (SI) engine operating at a rated speed of 4500 RPM was selected. The primary input parameters

for this analysis, including operational, geometric, and mass properties, were entered directly into the dedicated input panel. A user interface of the populated input panel is shown in Figure 4.2, which serves as the definitive record of the initial conditions for this case study.

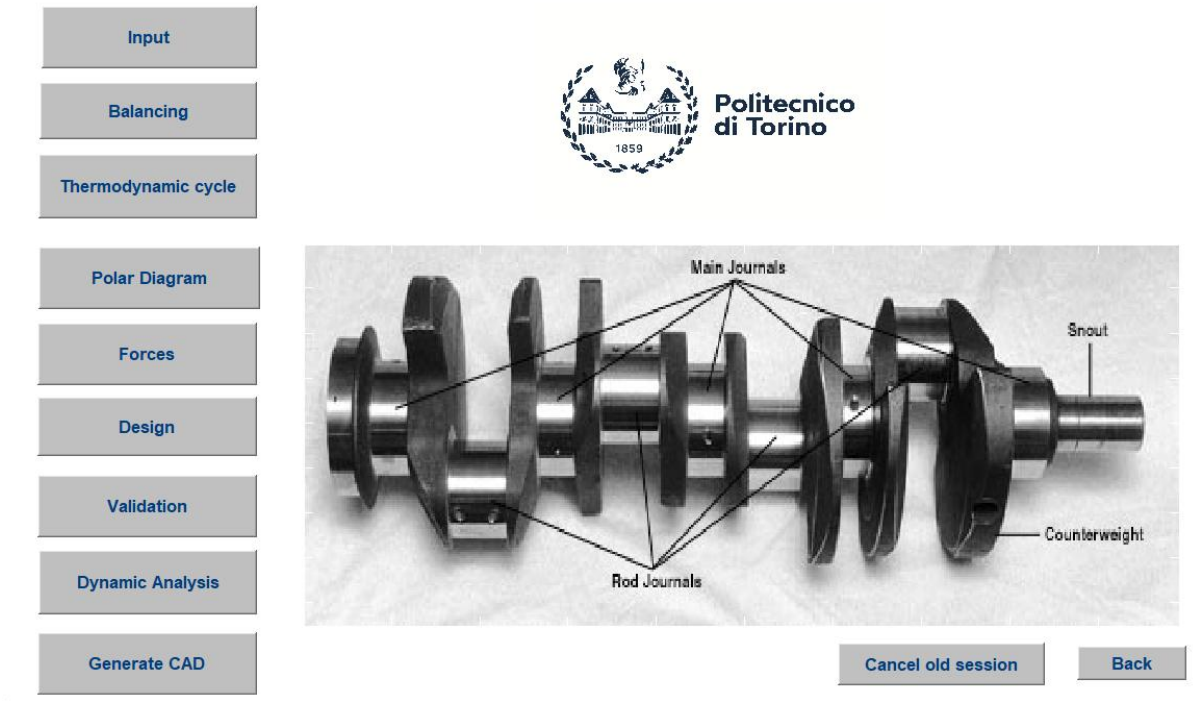

Figure 4.1: The main layout of the MATLAB GUI, showing the modular workflow.

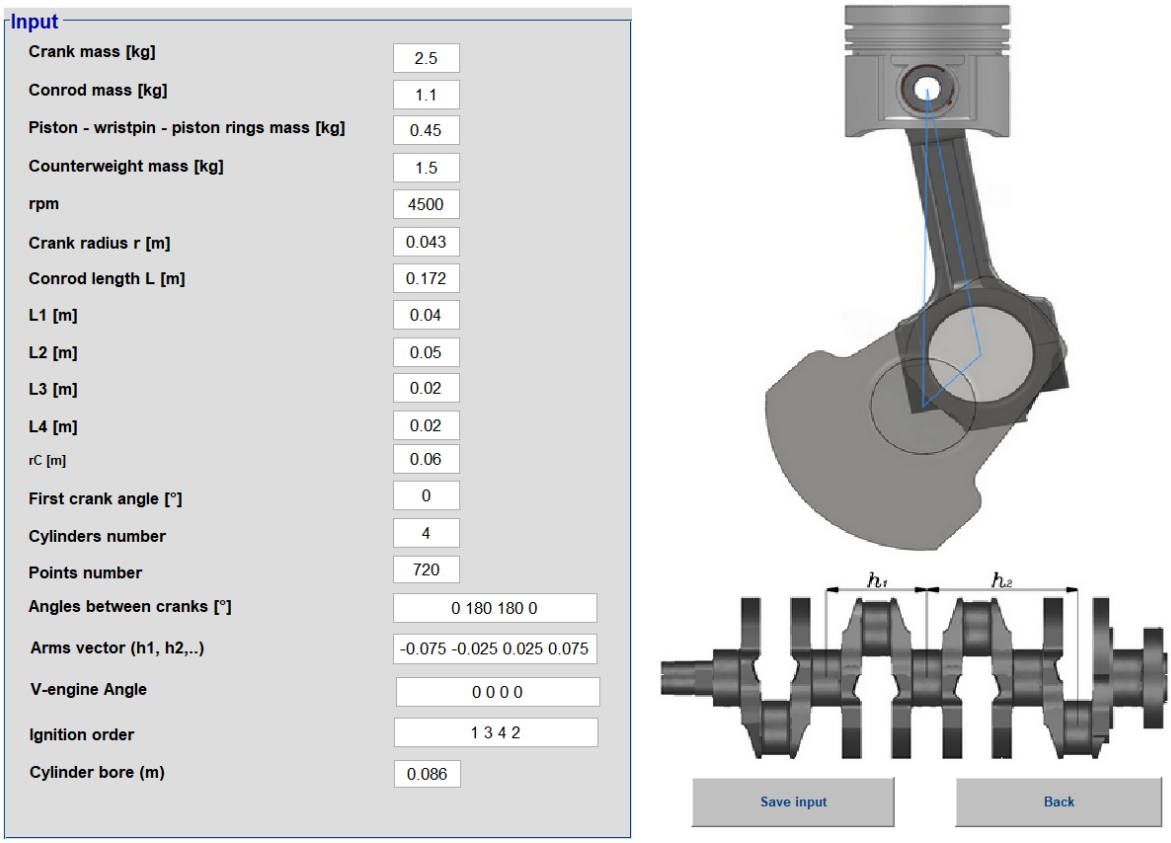

Figure 4.2: The populated 'Input' panel with the defined parameters for the case study.

4.2. Engine Balancing Analysis

Following the definition of the input parameters, the 'Balancing' module is executed. As confirmed by the underlying computational script, this stage of the analysis isolates and evaluates the forces and moments generated exclusively by the inertia of the moving components, independent of gas pressure effects. The framework first calculates the primary and secondary inertia force components (F_{x1}, F_{y1} and F_{x2}, F_{y2}) for the reciprocating and rotating masses of each individual cylinder.

These component vectors are then summed, considering the specific phase shifts dictated by the engine's crank angles (0-180-180-0) and firing order (1-3-4-2), to determine the resultant unbalanced forces and moments for the entire engine. The complete results from this module are displayed within a single 'Balancing' GUI layout, as shown in Figure 4.3. This screen presents a comprehensive overview of the engine's inherent balance characteristics, with Cartesian plots showing the x and y components of the resultant forces (R_{x1}, R_{y1}, R_{x2}, R_{y2}) and moments (M_{x1}, M_{y1}, M_{x2}, M_{y2}) over a full 720° engine cycle.

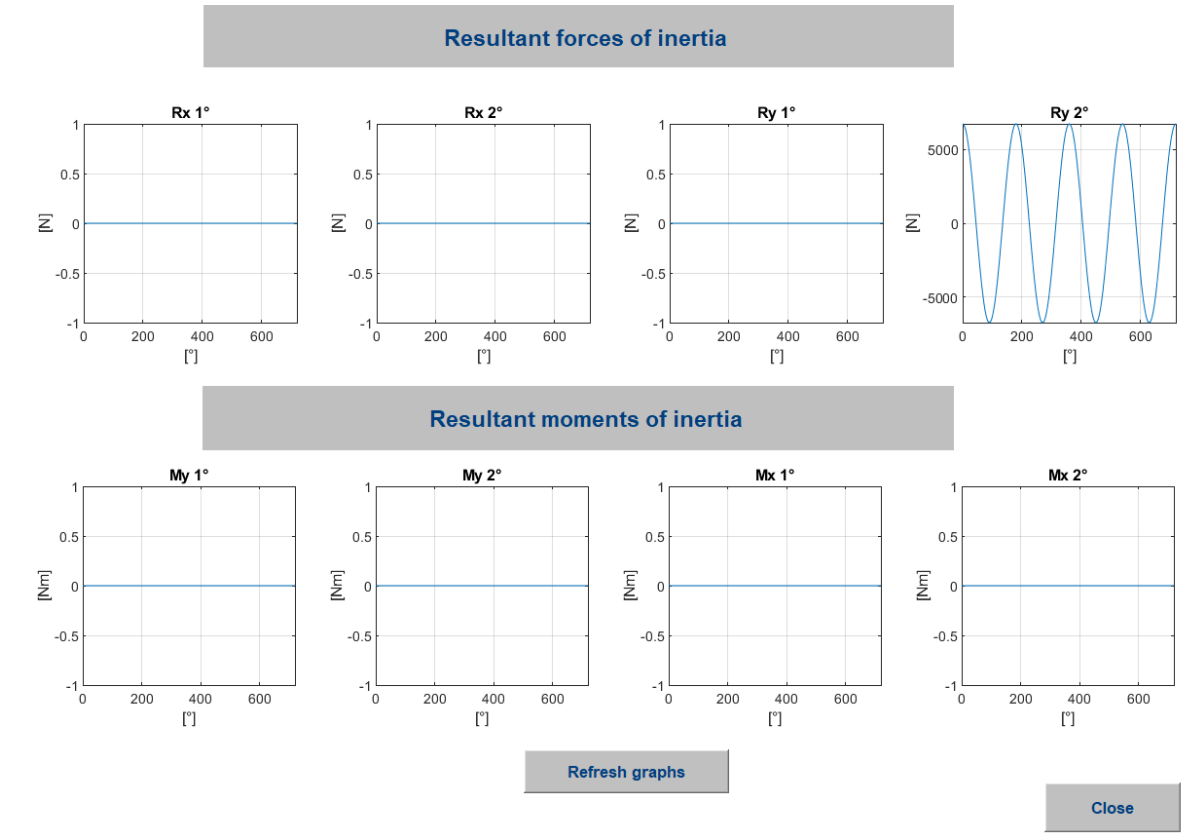

Figure 4.3: The 'Balancing' GUI, showing the Cartesian plots for the resultant primary and secondary inertia force and moment components vs. crank angle.

4.3. Thermodynamic Cycle Definition

The next step in the workflow is the 'Thermodynamic cycle' module, which defines the primary source of loading on the crankshaft: the in-cylinder gas pressure resulting from combustion. For this case study, the theoretical Otto cycle for a Spark Ignition (SI) engine was selected.

The framework's underlying script uses fundamental thermodynamic principles to compute the pressure profile based on the user-defined inputs shown in the GUI. The module calculates the pressure and volume for the entire 720° cycle, encompassing the compression, combustion, expansion, and exhaust strokes. The key output of this process is the pressure vector as a function of crank angle, which is visualized in the GUI as shown in Figure 4.4 and stored for subsequent use. This pressure data serves as the critical input for all subsequent force and torque calculations.

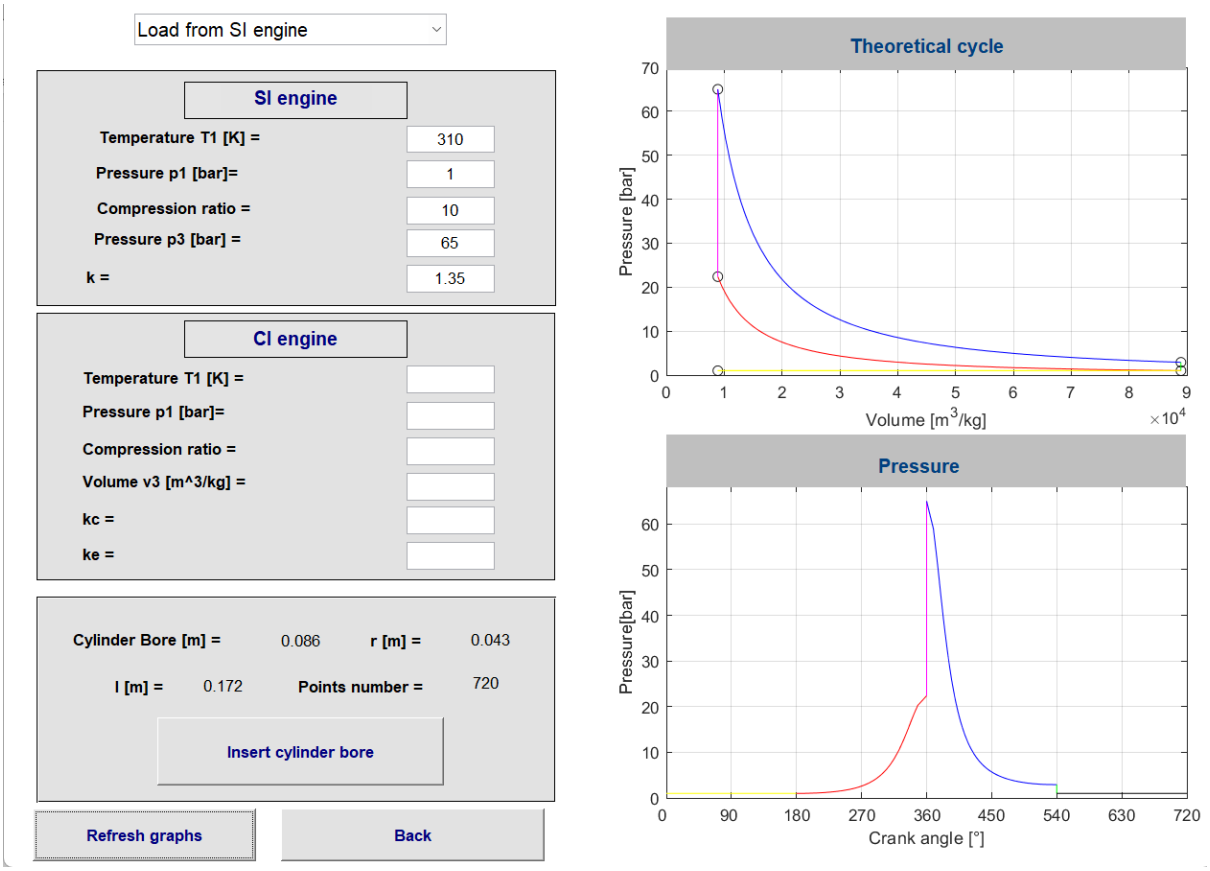

Figure 4.4: The 'Thermodynamic cycle' GUI, showing the user-defined parameters and the resulting pressure vs. crank angle diagram for the theoretical Otto cycle.

4.4. Polar Diagram and Total Force Analysis

With both the inertia forces and gas pressure defined, the 'Polar Diagram' module is used to compute and visualize the total combined load acting on the crankpin. The framework calculates the gas force vector for each cylinder from the pressure curve and sums it with the corresponding inertia force vector at each increment of the crank angle. This produces the net resultant force, which represents the true load experienced by the crankpin. The primary results are presented in a dedicated GUI, as shown in Figure 4.5, which provides two distinct visualizations for a user-selected cylinder:

- **Polar Diagram:** A polar plot showing the magnitude and direction of the total resultant force over a 720° cycle. This is a standard industry tool for visualizing the cyclical loading pattern on the bearing.

- **Force Hodograph:** A Cartesian plot of the resultant force's y-component versus its x-component. This diagram illustrates the path traced by the tip of the force vector, providing a clear depiction of the load's variation in magnitude and direction.

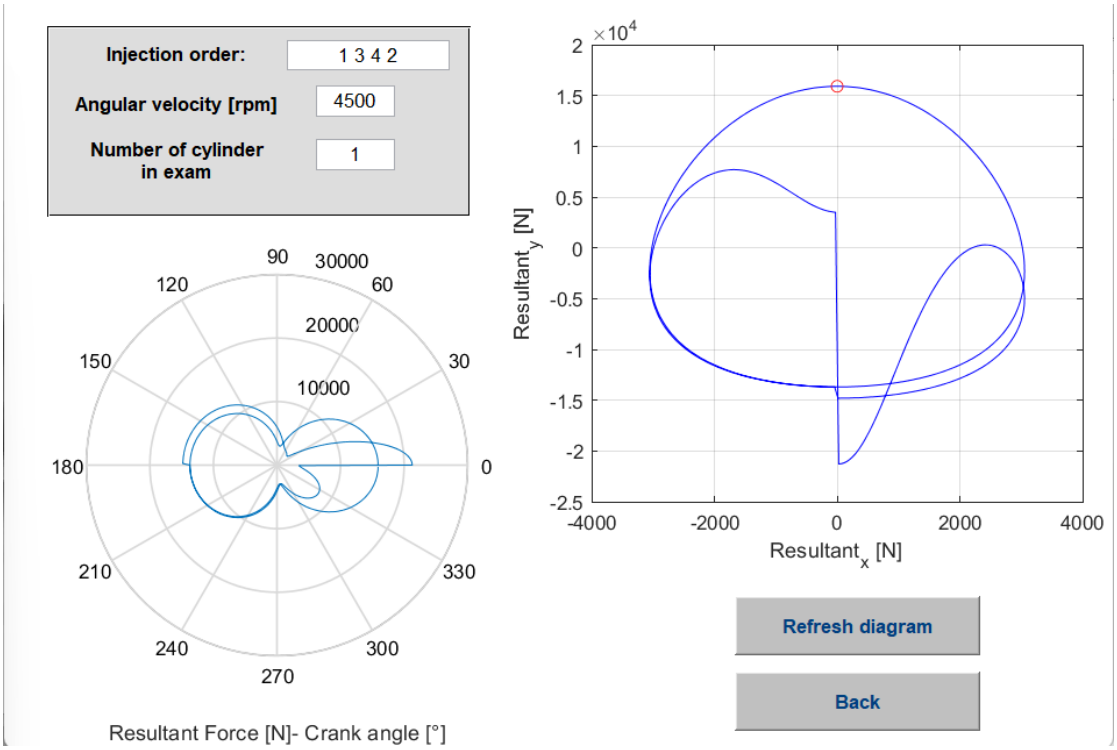

Figure 4.5: The 'Polar Diagram' GUI, showing the polar plot and the Cartesian force hodograph for a selected cylinder.

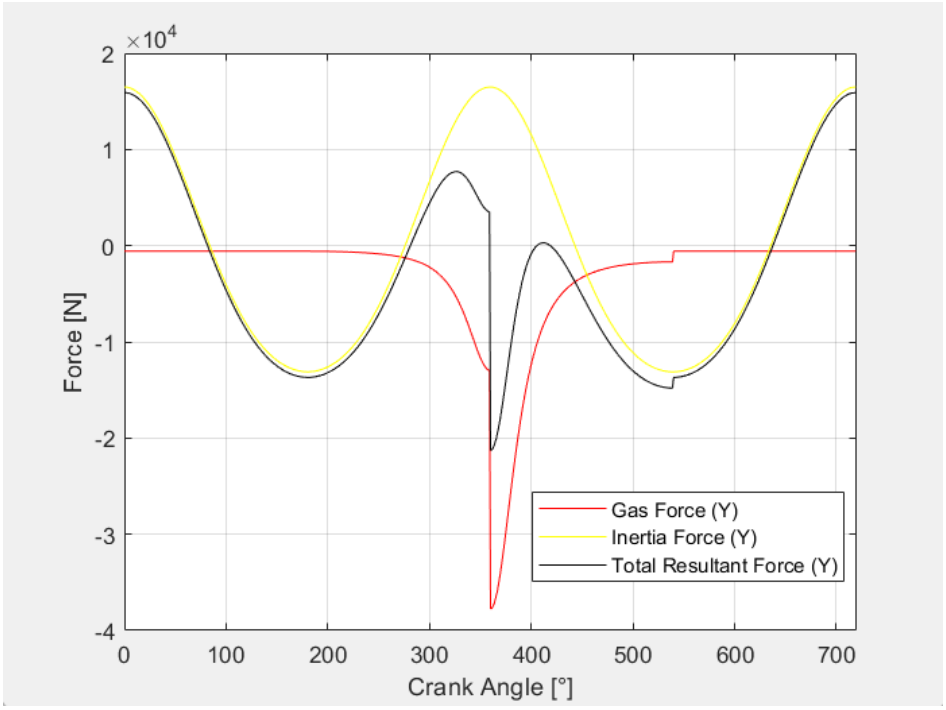

Figure 4.6: Plot showing the y-components of the gas force, inertia force, and total resultant force vs. crank angle.

In addition to the embedded visuals, the module generates a separate, detailed graph, as shown in Figure 4.6. This plot provides a clear decomposition of the y-components of the gas force and inertia force, as well as the total resultant force, plotted against the crank angle. This allows for a more granular analysis of how each component contributes to the overall load profile.

4.5. Force and Torque Component Analysis

Subsequent to the analysis of the resultant force, the 'Forces' module is utilized to decompose this force into its fundamental components and visualize the resulting engine torque. The framework resolves the total resultant force vector on the crankpin into its tangential and radial components, which are critical for subsequent stress and performance analysis.

The results are displayed in a dedicated GUI, as shown in Figure 4.7. The interface presents three key plots:

- **Tangential Force:** A plot of the force component acting perpendicular to the crank throw. This force is directly responsible for generating the engine's output torque.

- **Radial Force:** A plot of the force component acting along the axis of the crank throw, which is transmitted through the connecting rod to the main bearings.

- **Single-Cylinder Torque:** A plot of the instantaneous torque generated by the tangential force of a single cylinder over a complete engine cycle. While the torque fluctuates, the average value of this curve provides the mean torque, which is a fundamental measure of the engine's power output.

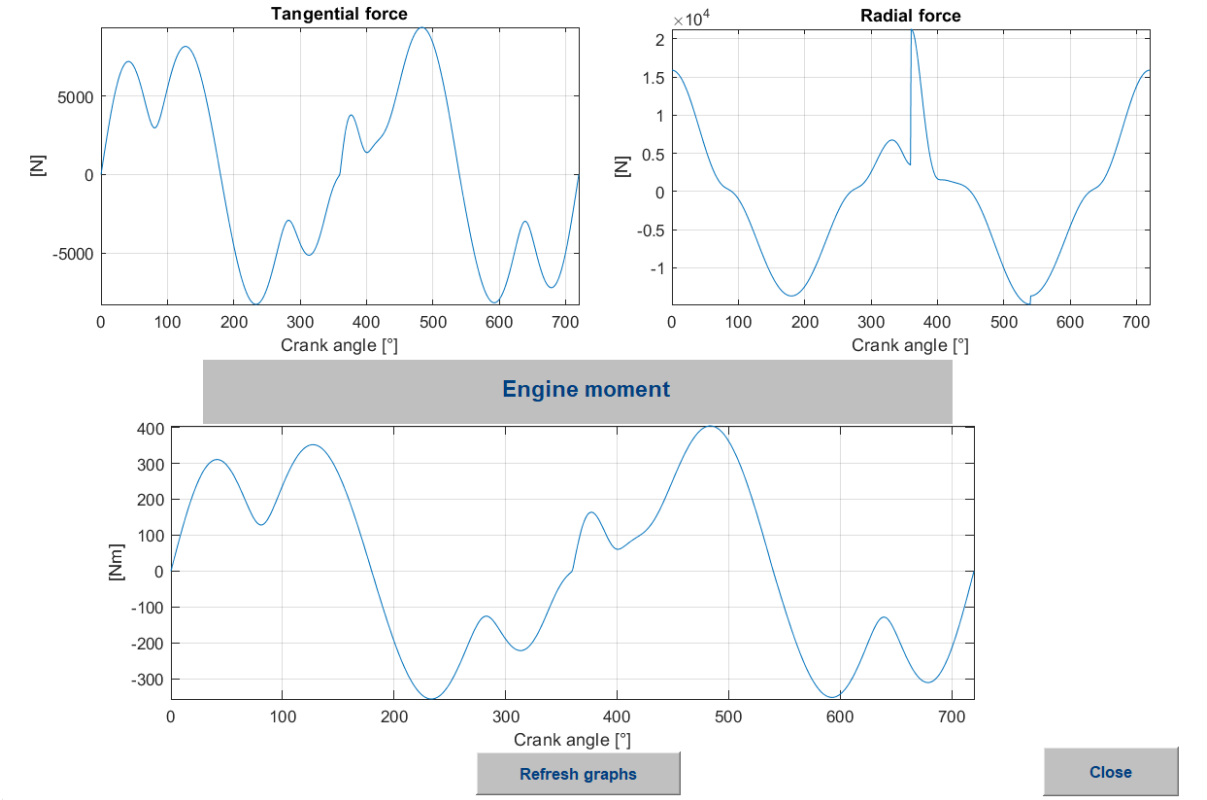

Figure 4.7: The 'Forces' GUI, displaying the tangential force, radial force, and single-cylinder torque plots versus crank angle.

4.6. Preliminary Design and Structural Verification

The 'Design' module translates the analytical forces into a physical design and performs a critical structural verification. This stage begins with the user selecting a material for the crankshaft. Based on the material's properties and the engine's peak pressure, the framework's underlying script calculates an initial set of geometric dimensions for the crankpins, journals, and webs using established empirical engineering formulas.

These initial dimensions are then subjected to a rigorous stress verification. The module identifies the points of maximum load and calculates the combined bending and torsional stresses at these critical locations. The calculated stress is then compared against the selected material's allowable stress limit to provide a clear, binary validation status. For this initial run of the case study, the empirically derived dimensions resulted in calculated stresses exceeding the material's limit, yielding a "NOK" (Not OK) status, as shown in Figure 4.8. This result is crucial as it demonstrates the framework's ability to identify a non-viable design before

proceeding to more complex analyses. The GUI also presents the minimum required diameters calculated from fatigue analysis, providing a clear target for the design iteration that will be performed in the subsequent 'Validation' stage.

To help diagnose the source of the high loads, the module includes a feature to analyse the engine's torque profile in greater detail. This opens a separate interface containing buttons to generate two distinct plots. The first plot, shown in Figure 4.9, displays the individual torque contribution from each of the four cylinders, allowing for an analysis of each cylinder's performance. The second plot, shown in Figure 4.10, displays the cumulative, total engine torque as each cylinder fires in sequence. This latter plot is essential for understanding how the torque pulses from each cylinder combine to produce the peak total torque that the crankshaft must withstand.

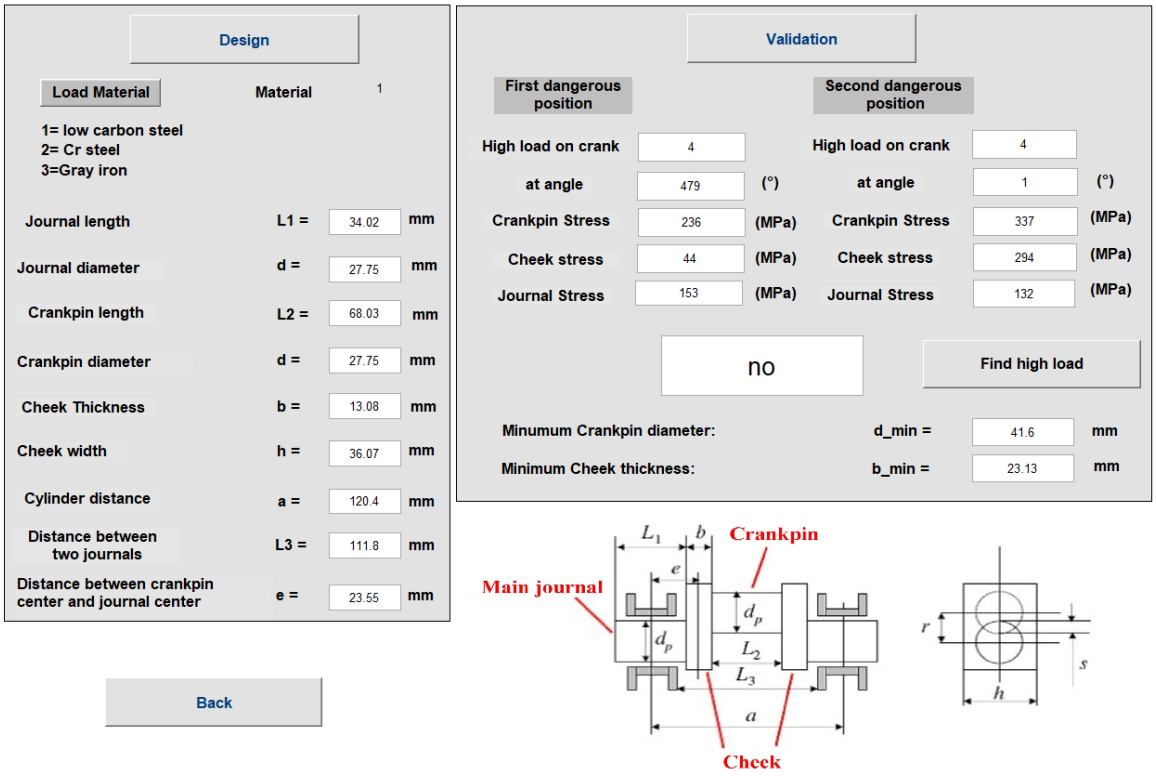

Figure 4.8: The 'Design' GUI, showing the selected material, the empirically calculated initial dimensions, and the resulting "NOK" stress verification status.

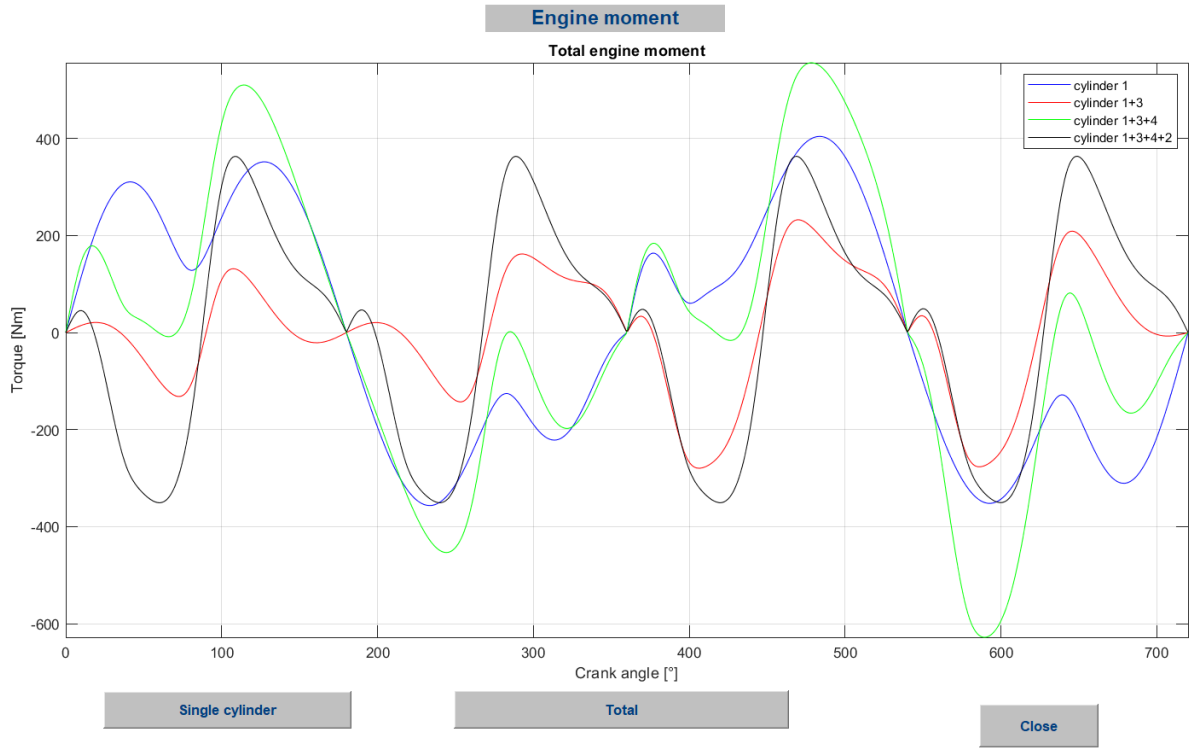

Figure 4.9: Plot of the individual torque contribution from each cylinder versus crank angle.

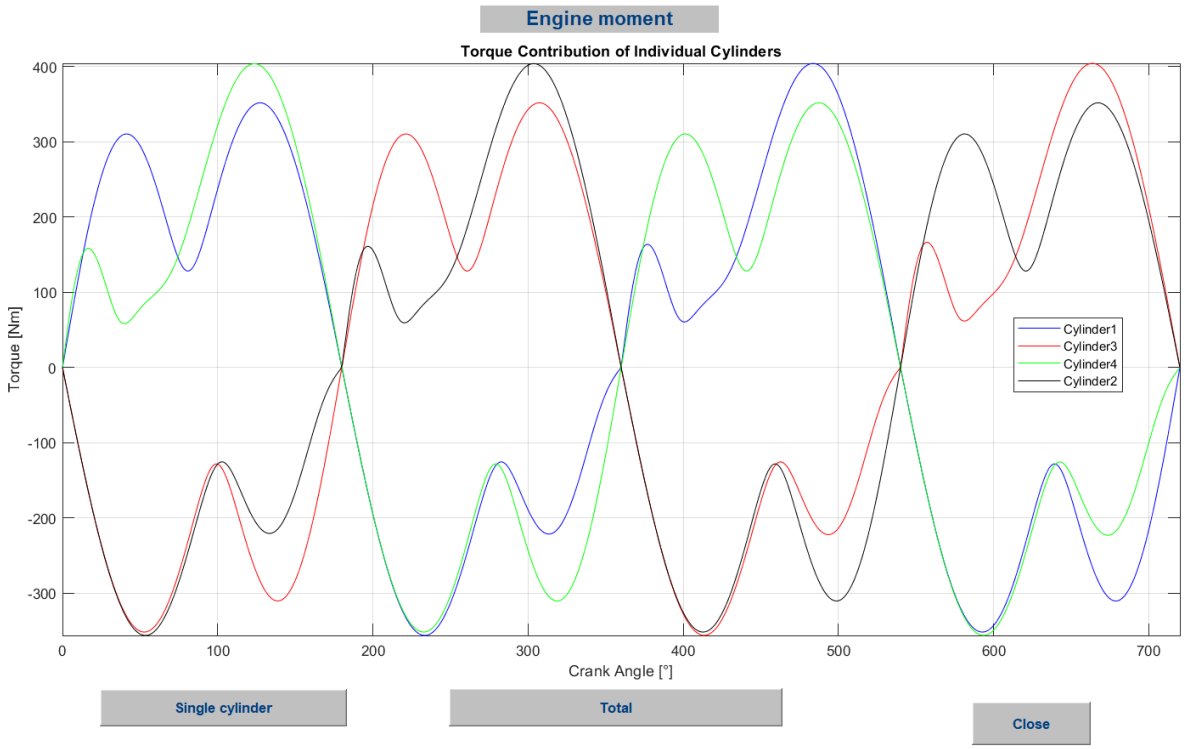

Figure 4.10: Plot of the cumulative total engine torque versus crank angle.

4.7. Design Validation and Iteration

The 'Validation' module represents the critical interactive and iterative phase of the design process, allowing the user to refine the initial, empirically derived geometry into a structurally viable component. The preceding 'Design' stage identified that the initial dimensions were insufficient, resulting in a "NOK" status and providing the user with calculated minimum required diameters for fatigue life.

Using these targets as a guide, the user can now directly edit the geometric parameters within the 'Validation' interface. The framework facilitates a real-time feedback loop: any modification to a dimension instantly triggers a recalculation of the combined stresses. This allows the user to iteratively adjust the crankpin diameter, web thickness, and other key parameters until the calculated stress falls below the material's allowable limit.

For the case study, the dimensions were incrementally increased until a successful validation was achieved, indicated by the status changing to "OK", as shown in Figure 4.11. This final set of dimensions represents a baseline design that is confirmed to be structurally sound under the maximum calculated loads. Throughout this iterative process, the user retains access to the same detailed torque analysis plots for both individual cylinders and the total engine (previously shown in Figures 4.9 and 4.10), allowing for a constant and comprehensive understanding of the load conditions while refining the geometry.

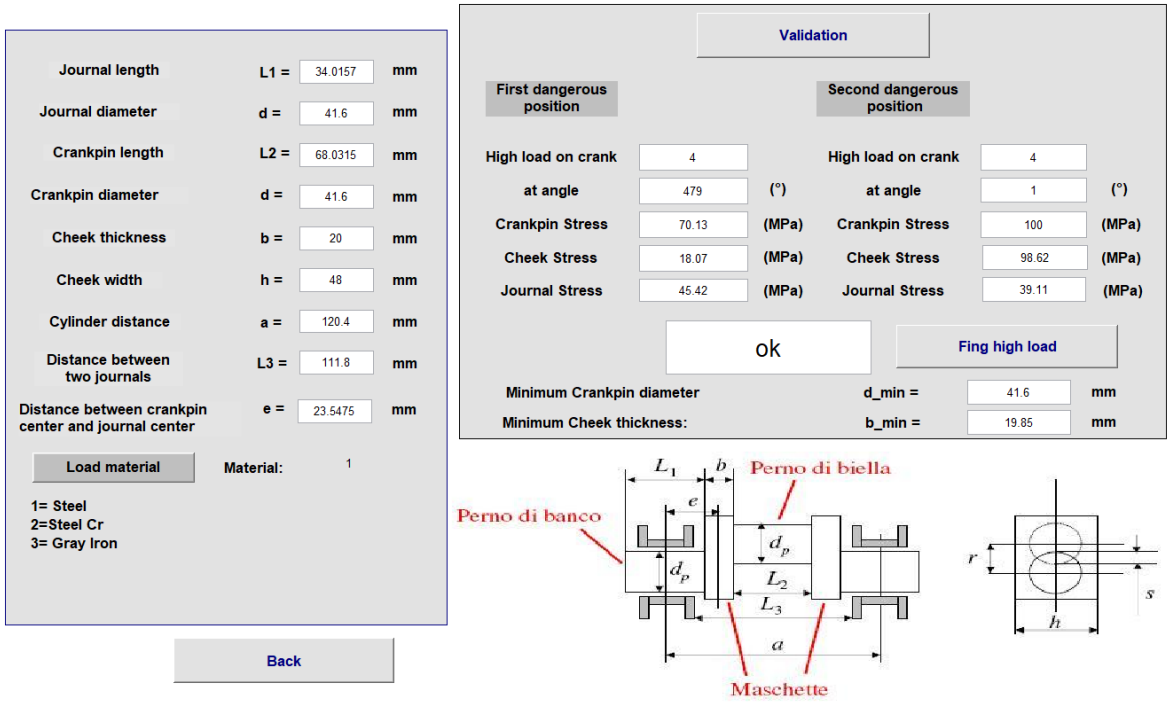

Figure 4.11: The 'Validation' GUI, showing the user-modified final dimensions and the successful "OK" stress verification status.

4.8. Dynamic and Vibrational Analysis

The final analytical stage is the dynamic analysis, which is crucial for ensuring the crankshaft's operational reliability by evaluating its torsional vibration characteristics. This complex process is managed through the 'Dynamic Analysis' GUI (Figure 4.12), which provides a comprehensive suite of tools for a staged investigation of the system's dynamic behaviour.

4.8.1. System Definition and Modal Analysis

The analysis workflow begins with the user defining the physical properties for the lumped-mass model. Once these parameters are set, the "Perform analysis" button is clicked. The framework programmatically constructs the global mass and stiffness matrices and solves the generalized eigenvalue problem to find the system's torsional natural frequencies and corresponding mode shapes.

The raw output from the eigenvalue solver is automatically processed by the framework to ensure a clear and physically intuitive presentation. The calculated natural frequencies are sorted in ascending order, and the corresponding mode shapes are normalized for consistent visualization before being populated in the GUI's text fields. To understand the physical nature of these resonances, the user can visualize the mode shapes by clicking the associated 'mode' buttons. Figures 4.13 through 4.17 show all five flexible modes of vibration for the system, illustrating the distinct patterns of torsional deformation along the crankshaft. The visualization of these mode shapes is more than just a graphical output; it is a critical validation of the underlying physical model. The distinct, physically realistic patterns of torsional deformation confirm that the mass and stiffness matrices were formulated correctly and that the eigenvalue solution is sound. This step provides the user with fundamental confidence in the system's dynamic properties before proceeding to the forced response analysis.

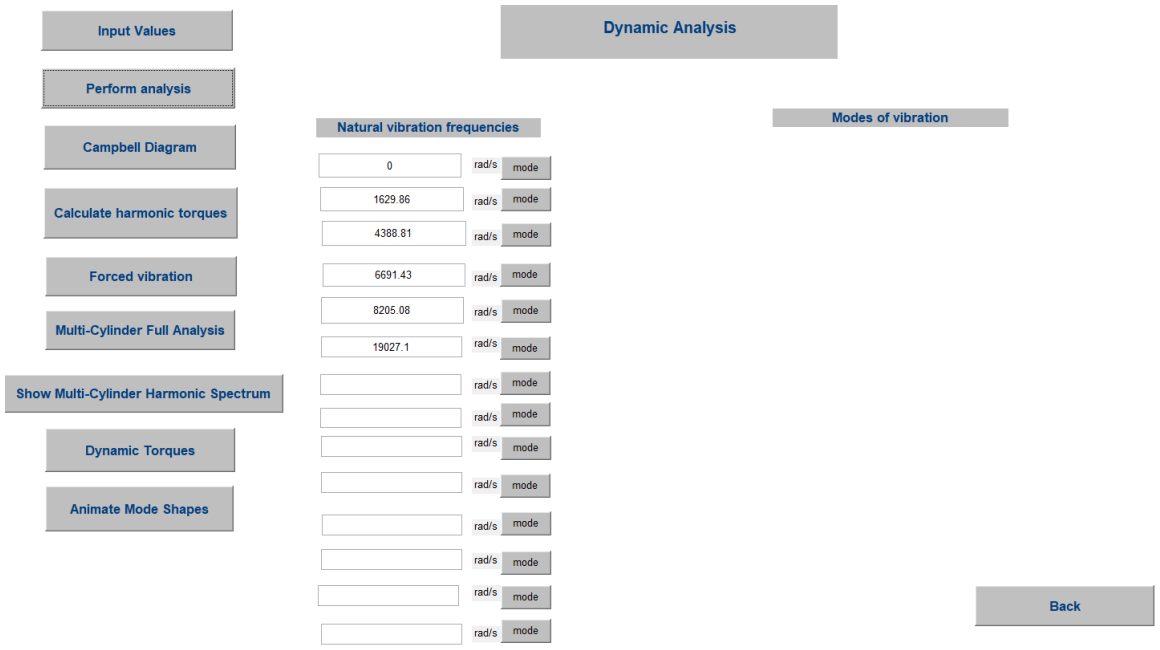

Figure 4.12: The 'Dynamic Analysis' GUI populated with calculated natural frequencies after performing the modal analysis.

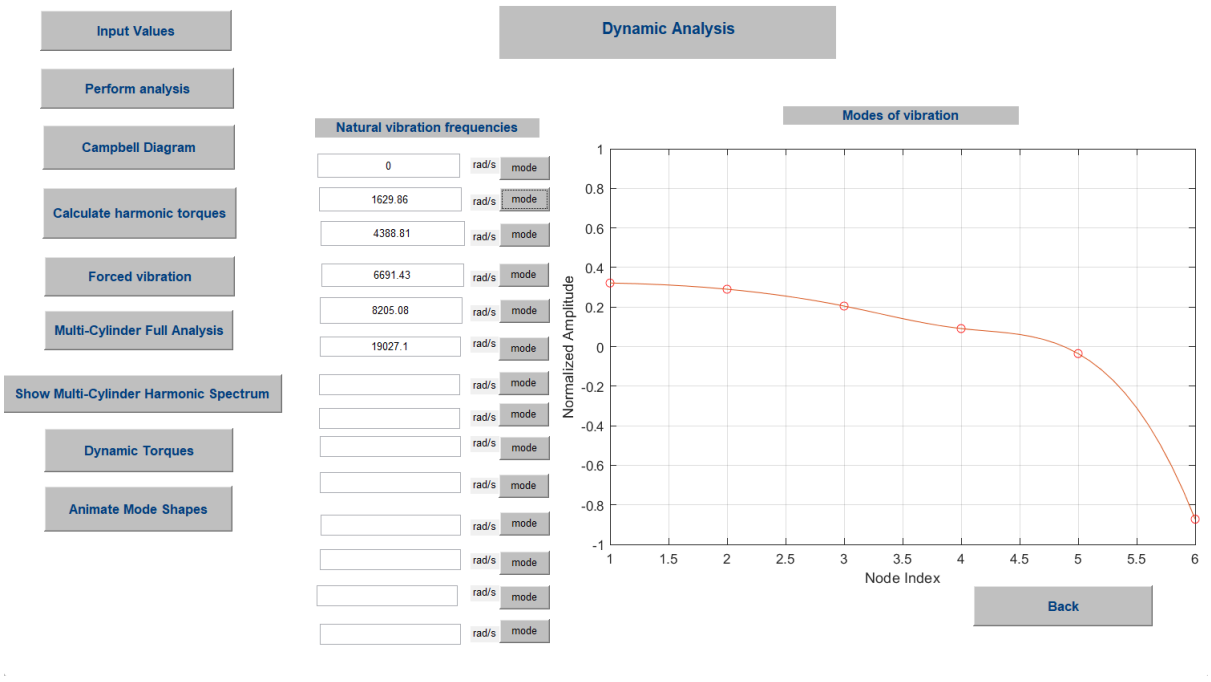

Figure 4.13: Visualization of the second torsional mode shape (first flexible mode).

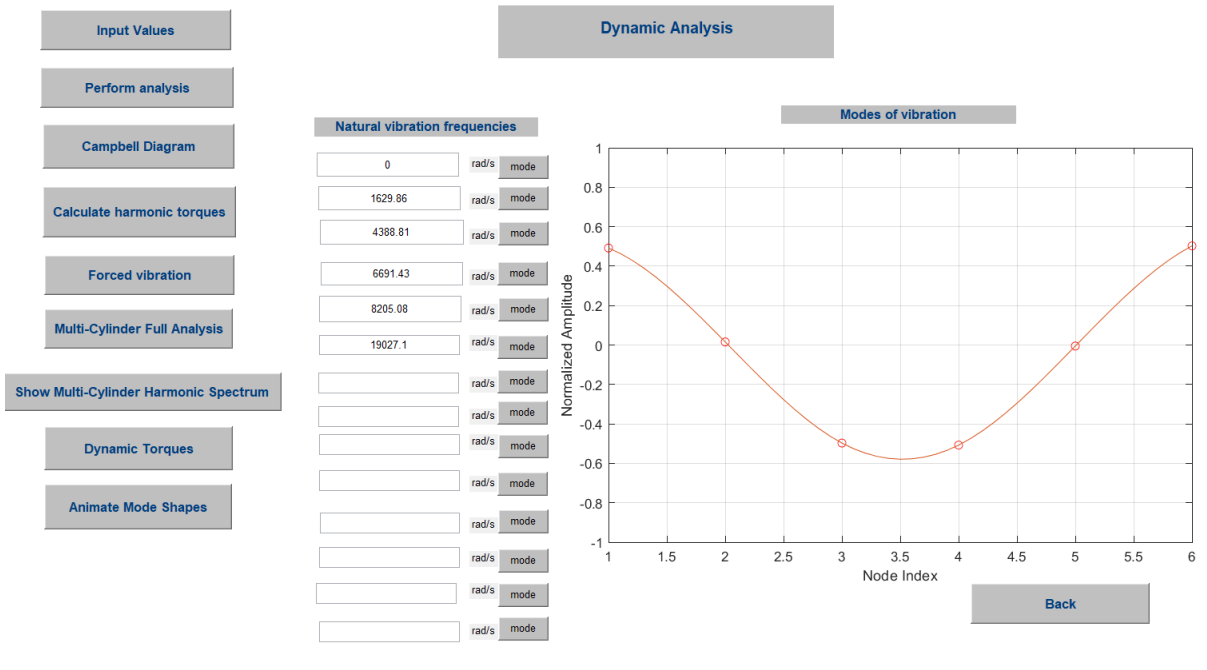

Figure 4.14: Visualization of the third torsional mode shape (second flexible mode).

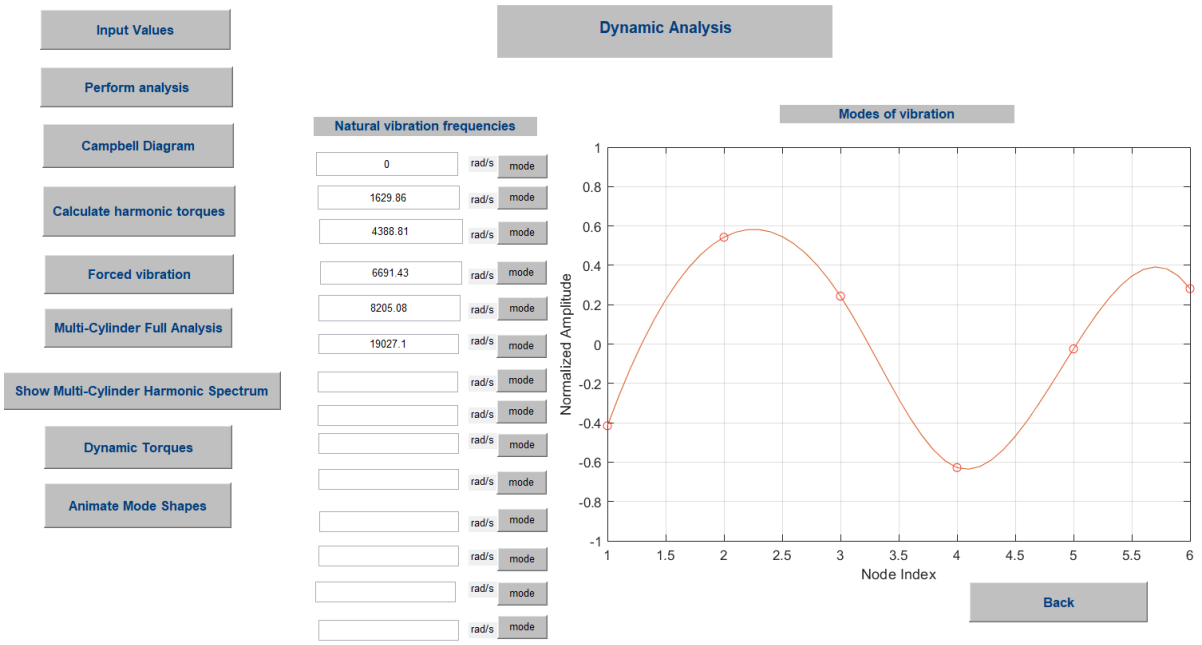

Figure 4.15: Visualization of the fourth torsional mode shape (third flexible mode).

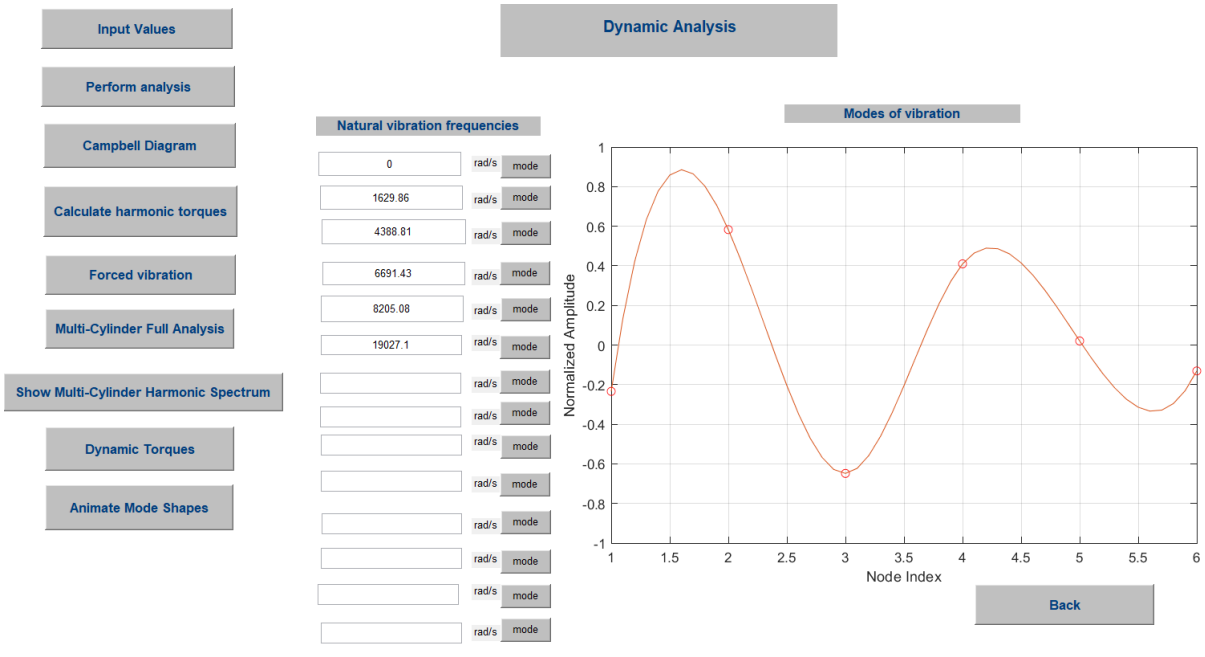

Figure 4.16: Visualization of the fifth torsional mode shape (fourth flexible mode).

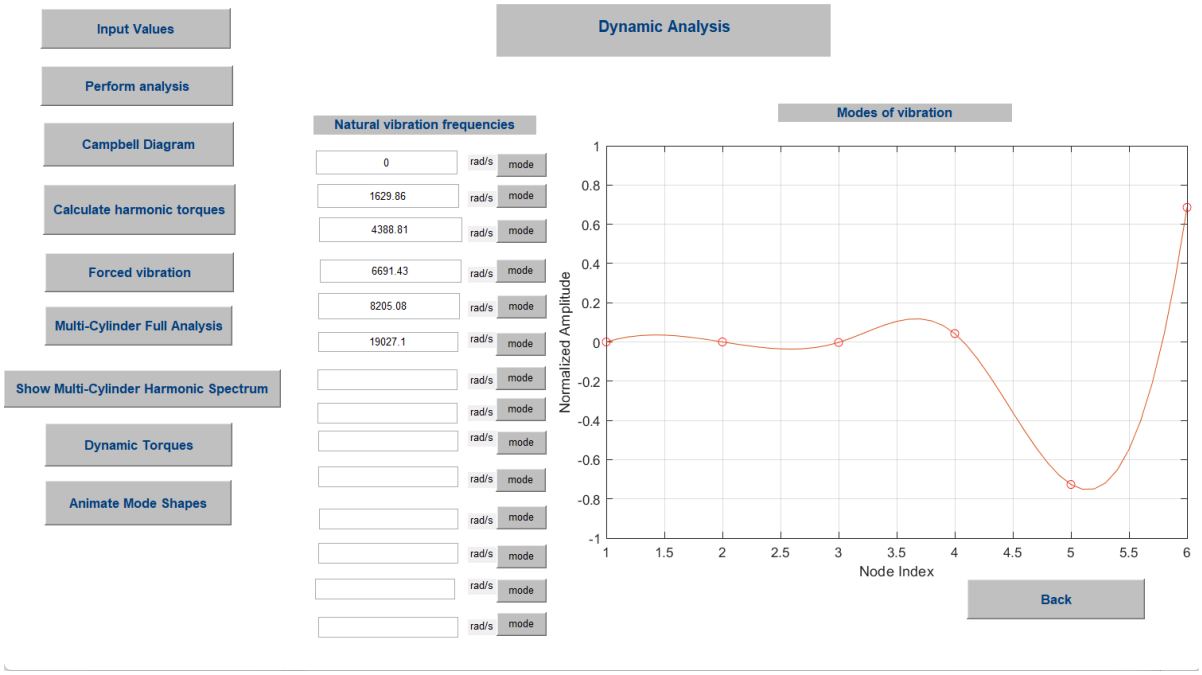

Figure 4.17: Visualization of the sixth torsional mode shape (fifth flexible mode).

4.8.2. Preliminary Resonance Identification and Single-Cylinder Analysis

Once the modal analysis is complete, the user can perform preliminary checks. The user first generates the Campbell Diagram (Figure 4.18). This critical tool plots the system's inherent natural frequencies against all potential engine excitation orders across the RPM range. The intersection points identify all potential "critical speeds." To supplement the dense visual data of the Campbell Diagram, the framework also presents a clear summary of the exact RPM and frequency for each identified critical speed. This quantitative data, shown in Table 4.1, ensures that no critical intersection is missed and provides the user with precise values for their final analysis.

Next, the user can perform a preliminary forced response analysis based on a simplified single-cylinder model by calculating the harmonic torques (Figure 4.19) and generating a preliminary vibration response curve (Figure 4.20). This staged approach is a key methodological strength. The Campbell Diagram provides an immediate, high-level risk assessment, validating the framework's predictive capabilities without requiring a full torque analysis. The subsequent single-cylinder analysis demonstrates the framework's utility for rapid initial checks, allowing a designer to quickly identify potential areas of concern before committing to the more computationally intensive full-system analysis.

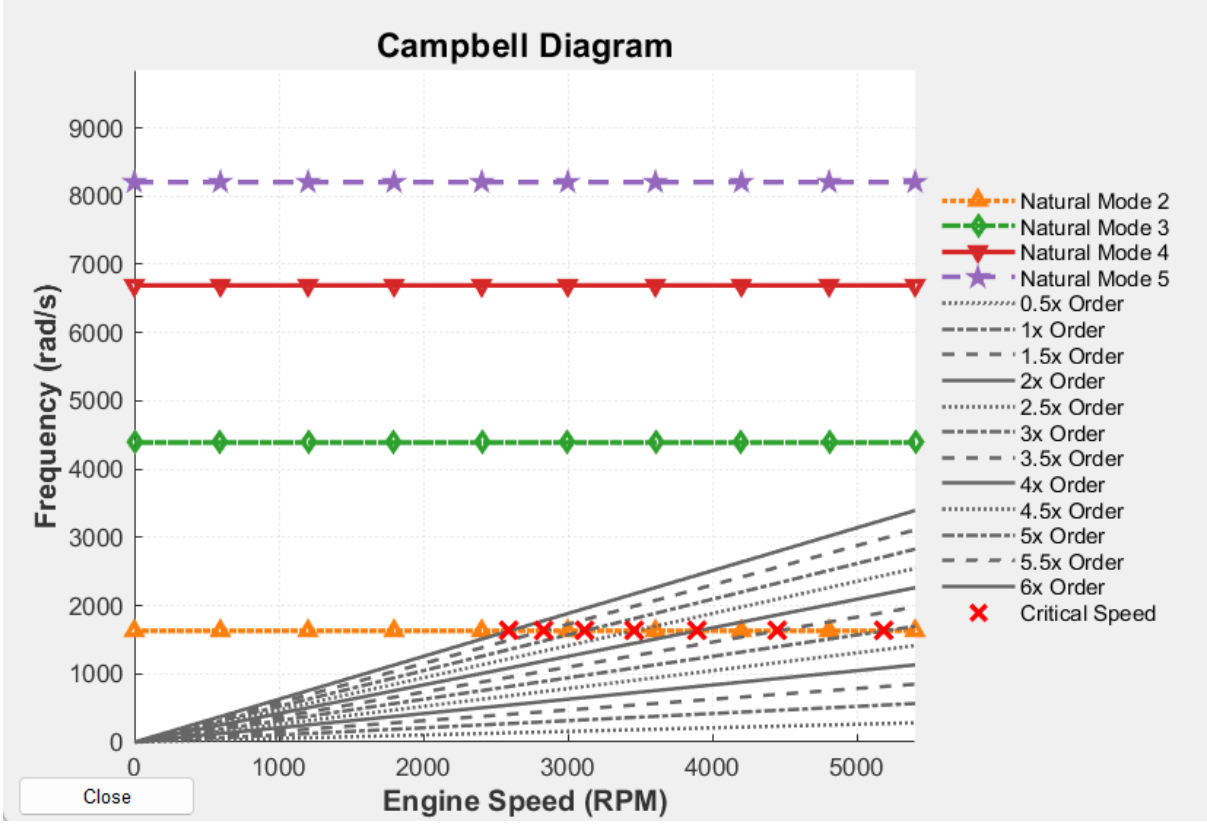

Figure 4.18: The Campbell Diagram, identifying potential critical speeds for the system.

MODE	Frequency (rad/s)	Order	Engine speed (RPM)
2	1629.86	6.0	2594
2	1629.86	5.5	2830
2	1629.86	5.0	3113
2	1629.86	4.5	3459
2	1629.86	4.0	3891
2	1629.86	3.5	4447
2	1629.86	3.0	5188

Table 4.1: Summary of Critical Speeds Identified from the Campbell Diagram

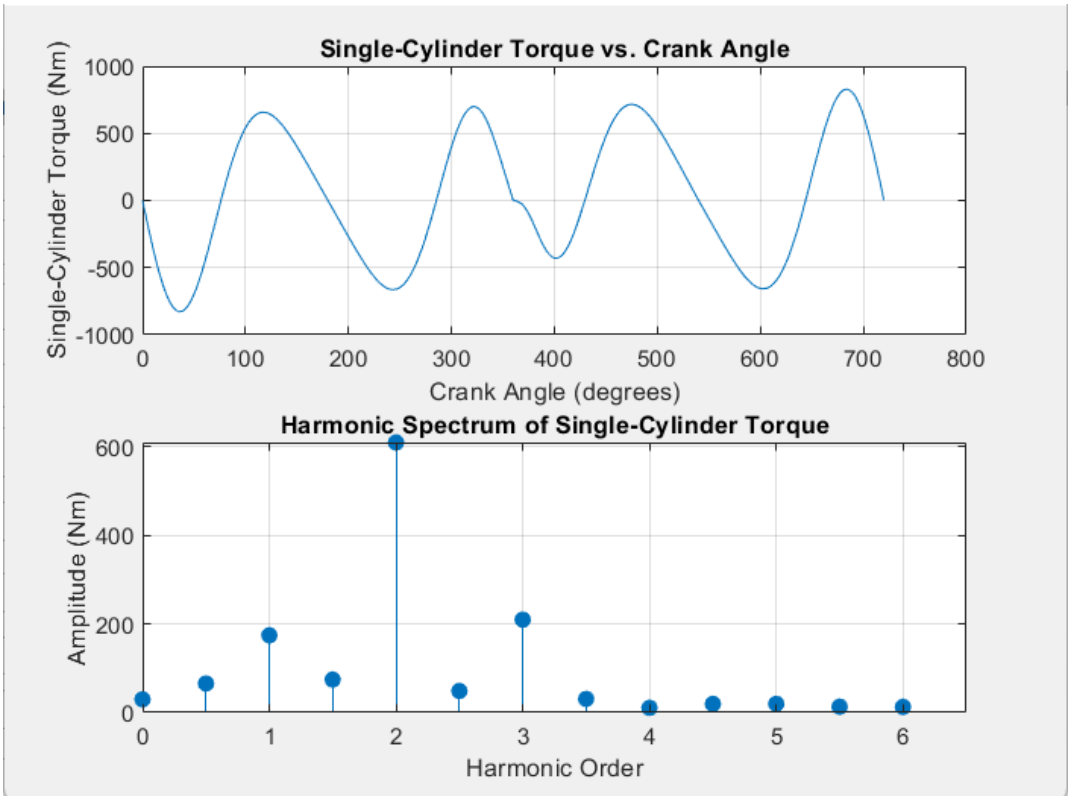

Figure 4.19: Harmonic spectrum of the single-cylinder torque profile.

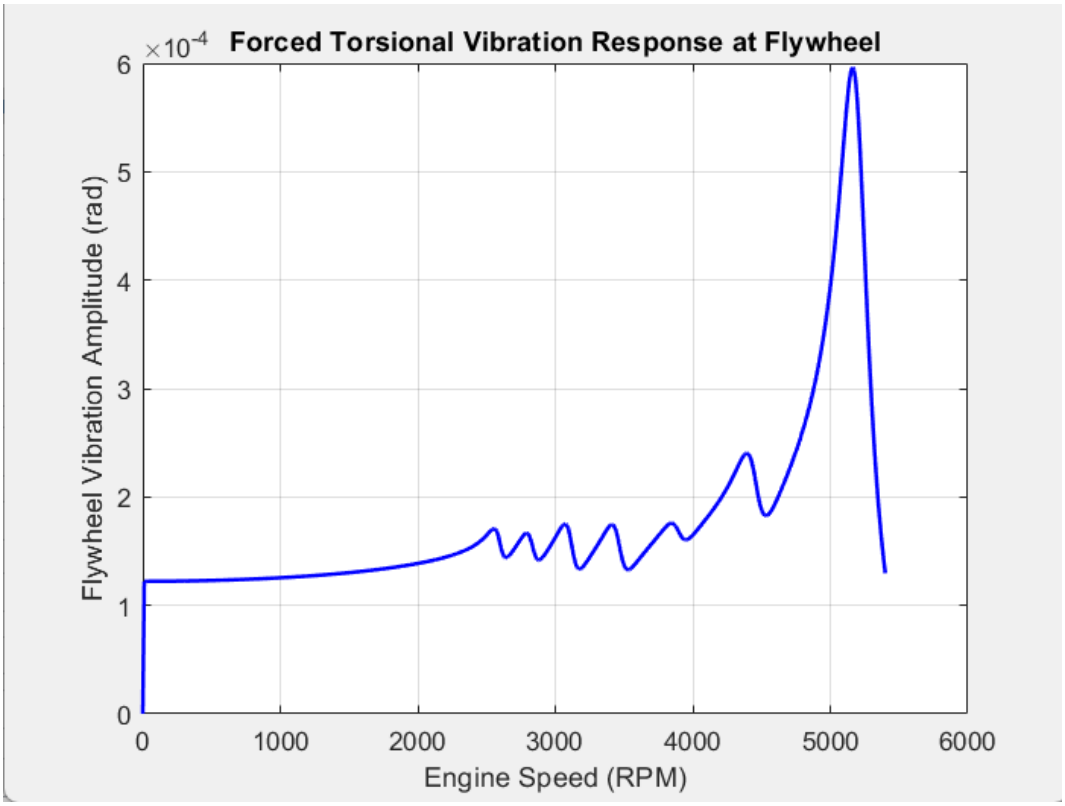

Figure 4.20: Preliminary forced vibration response at the flywheel based on the single-cylinder harmonic model.

4.8.3. Comprehensive Multi-Cylinder Analysis

The definitive evaluation is performed by clicking the "Multi-Cylinder Full Analysis" button. The framework calculates the true total engine torque from all cylinders and performs a new, more accurate Fourier analysis. The resulting multi-cylinder harmonic spectrum is shown in Figure 4.21. Using these accurate harmonics, the system then calculates the definitive forced vibration response, producing plots of the final vibration amplitude at the flywheel (Figure 4.22) and the dynamic torsional torques within each individual shaft section (Figure 4.23).

This final calculation represents the ultimate validation of the framework's dynamic analysis module. A comparison between the multi-cylinder harmonic spectrum (Figure 4.21) and the single-cylinder version (Figure 4.19) highlights the necessity of this comprehensive approach, as the true excitation profile is significantly different due to the constructive and destructive interference of torque pulses from all four cylinders. The final response plots (Figures 4.22 and 4.23) provide the most accurate quantitative assessment of the crankshaft's dynamic performance, confirming that the framework can deliver the high-fidelity results required for a final design sign-off.

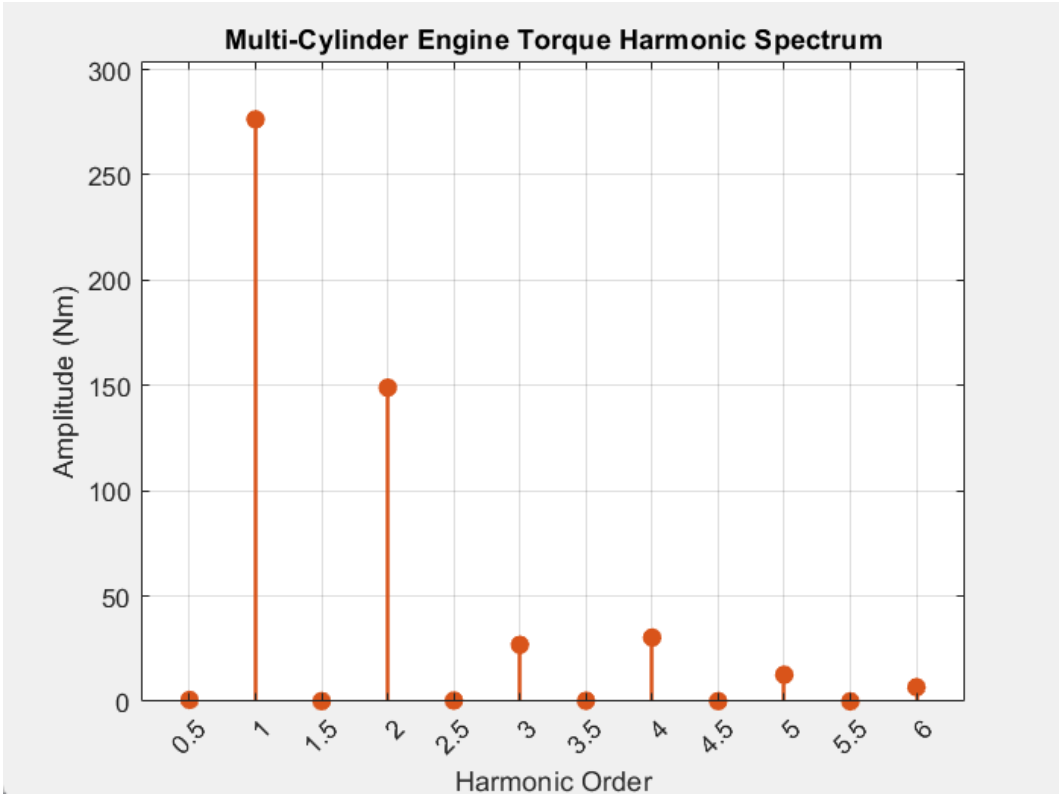

Figure 4.21: The multi-cylinder engine torque harmonic spectrum.

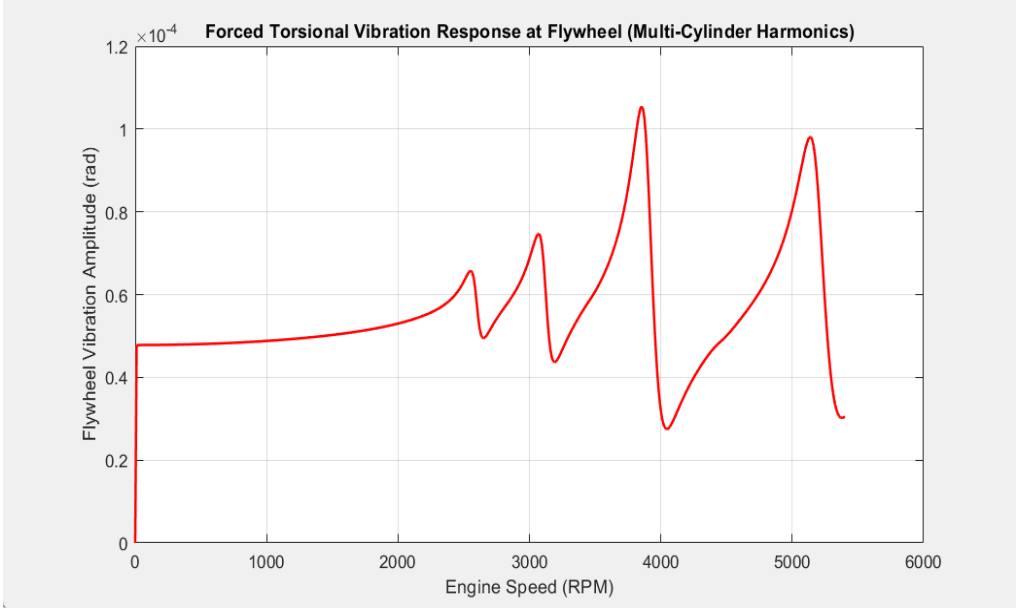

Figure 4.22: Final forced torsional vibration response at the flywheel based on multi-cylinder excitation.

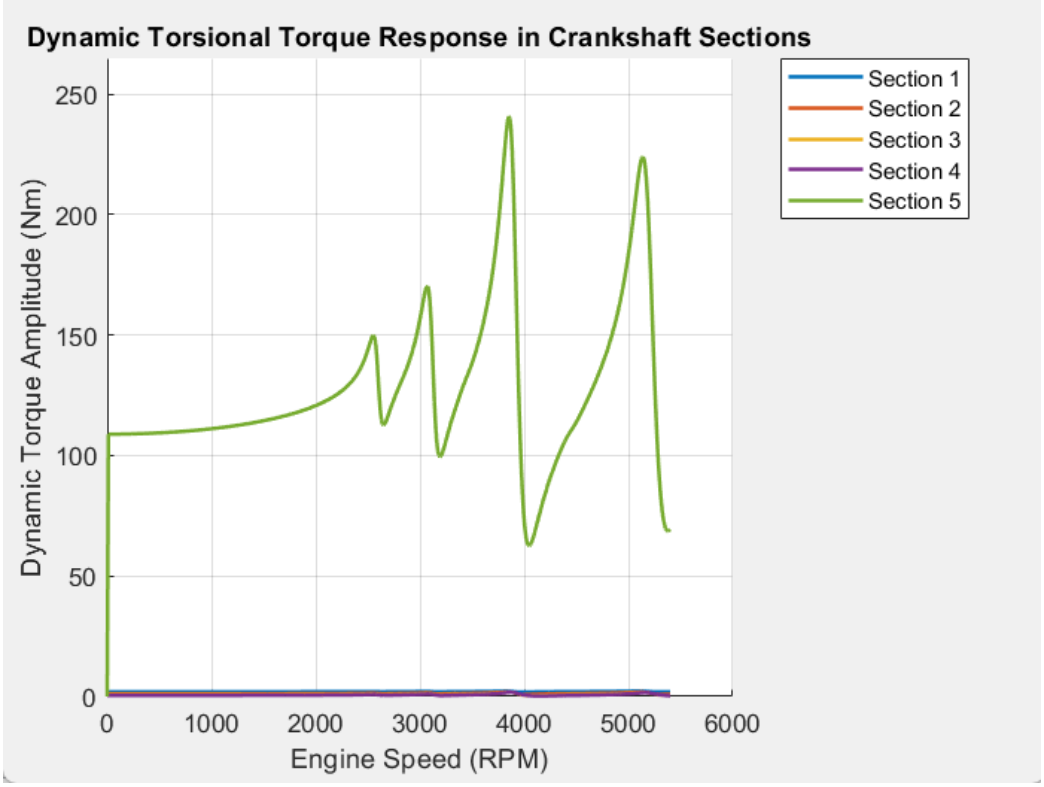

Figure 4.23: Dynamic torsional torque in each crankshaft section versus engine speed.

4.8.4. Mode Shape Animation

Finally, the "Animate Mode Shapes" button provides a qualitative and intuitive understanding of the system's dynamic behaviour by opening a new window (Figure 4.24) where the user can visualize an animated representation of the crankshaft twisting at each natural frequency. The value of this feature lies in its ability to bridge the gap between abstract numerical data and physical intuition. It makes the complex concept of torsional deformation accessible and understandable for the designer, validating the framework's focus not only on computational accuracy but also on user-centric design and effective data communication.

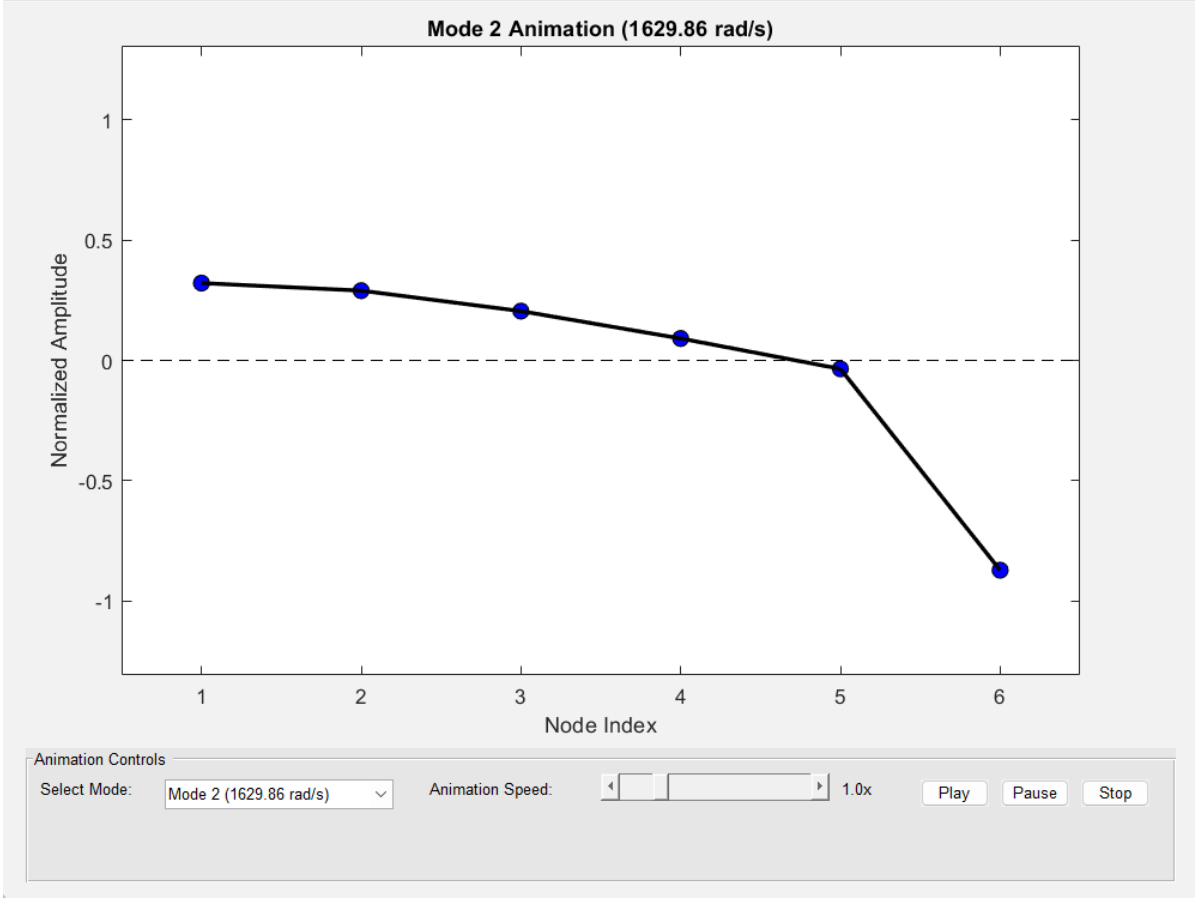

Figure 4.24: The mode shape animation interface, providing a visual representation of torsional deformation.

4.9. Automated CAD Model Generation

The culmination of the framework's integrated workflow is the automated generation of the crankshaft's 3D CAD model. This final stage serves as the ultimate validation of the end-to-end process, demonstrating the successful translation of abstract engineering requirements and complex analytical data into a tangible, geometrically defined component. The process utilizes a robust, file-based routine that bridges the computational environment of MATLAB with the geometric construction capabilities of SolidWorks.

The process is initiated from the MATLAB GUI by clicking the "Generate CAD" button. The framework first presents the user with a dialog box to confirm if fillets should be included in the final design parameters (Figure 4.25). Following this, a final instruction panel (Figure 4.26) is displayed, guiding the user on the subsequent steps required within the SolidWorks environment. The framework then automatically launches the SolidWorks application, seamlessly bridging the two software packages.

Once in SolidWorks, the automation is triggered via a custom macro, activated by the "Generate Crankshaft" button integrated into the user's toolbar, as highlighted in Figure 4.27. Activating the macro first opens a universal crankshaft template part. This template is pre-linked to the external equations file generated by MATLAB, causing it to immediately ingest the final, verified parameters from the analysis. The successful population of the SolidWorks Equation Manager, shown in Figure 4.28, provides a direct and objective validation of the seamless data transfer from the analytical framework to the CAD environment.

Following the parametric update, the macro programmatically executes a loop based on the number of cylinders, procedurally replicating and positioning the crank throws and journals into their correct orientations. The final output of the automated sequence is a single, monolithic solid part representing the core geometry of the crankshaft (Figure 4.29).

Following the user instructions, the final manual step involves applying fillets to the journal and crankpin edges. The required fillet radii, parametrically defined by the values exported from MATLAB, ensure design consistency. A detailed view of the final applied fillets at a critical crankpin-web junction is shown in Figure 4.30. This final, detailed geometry confirms

the framework's ability to not only perform a comprehensive engineering analysis but also to deliver a complete design baseline.

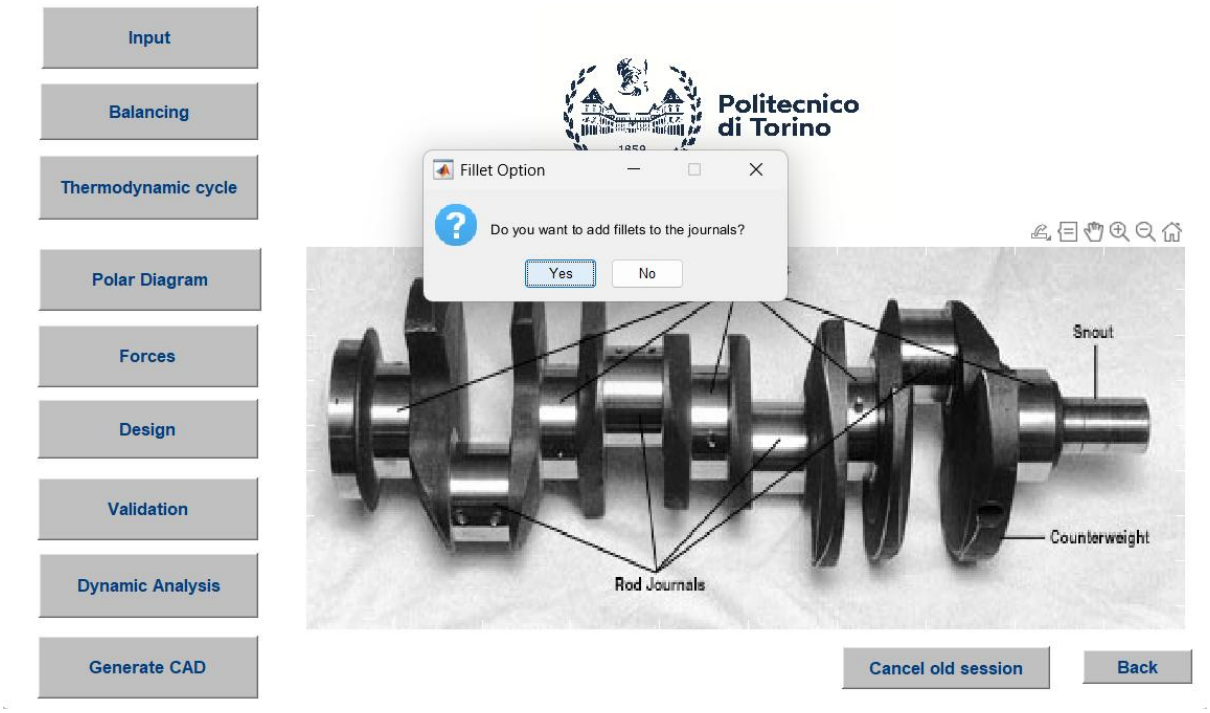

Figure 4.25: The MATLAB dialog box prompting the user to confirm the inclusion of fillets.

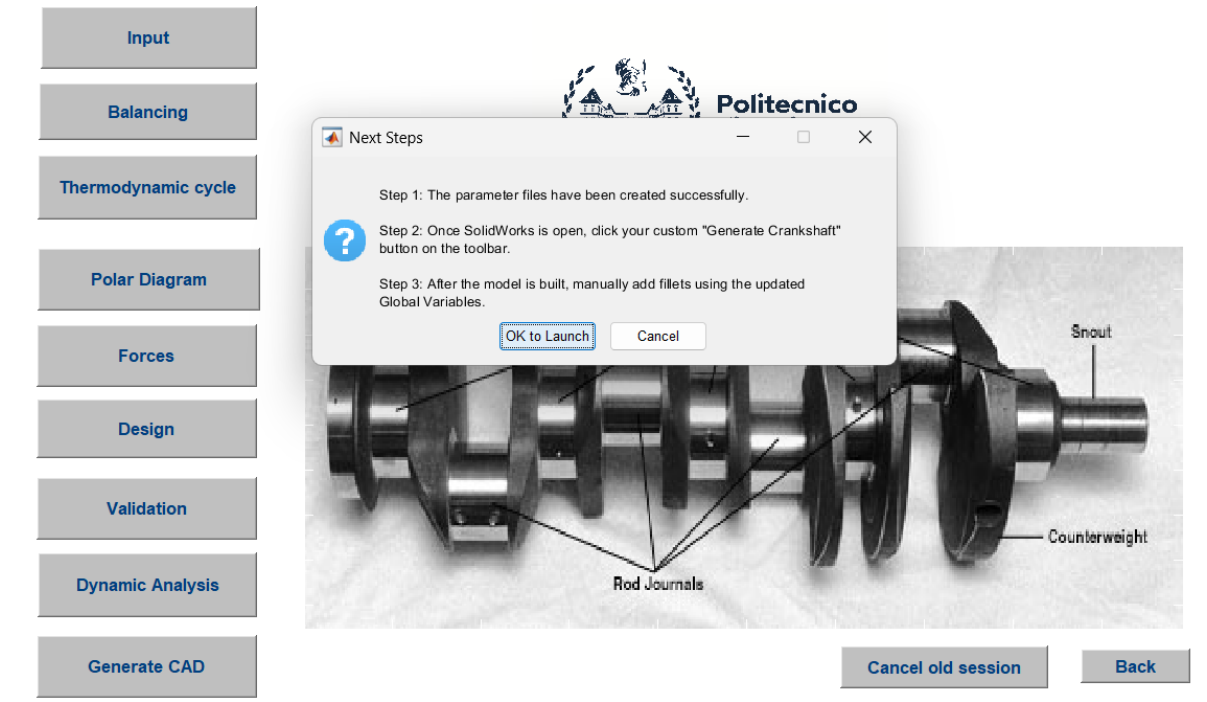

Figure 4.26: The MATLAB instruction panel displayed to the user after the data files have been generated.

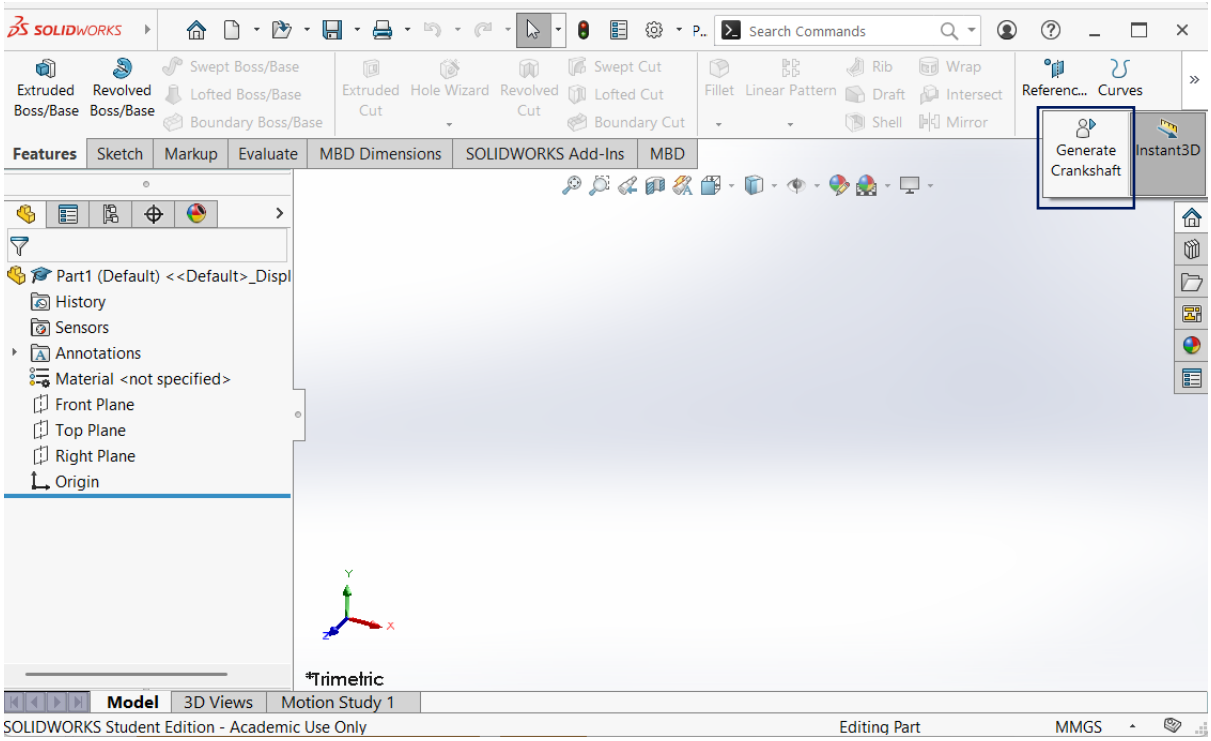

Figure 4.27: The custom "Generate Crankshaft" macro button highlighted in the SolidWorks user interface.

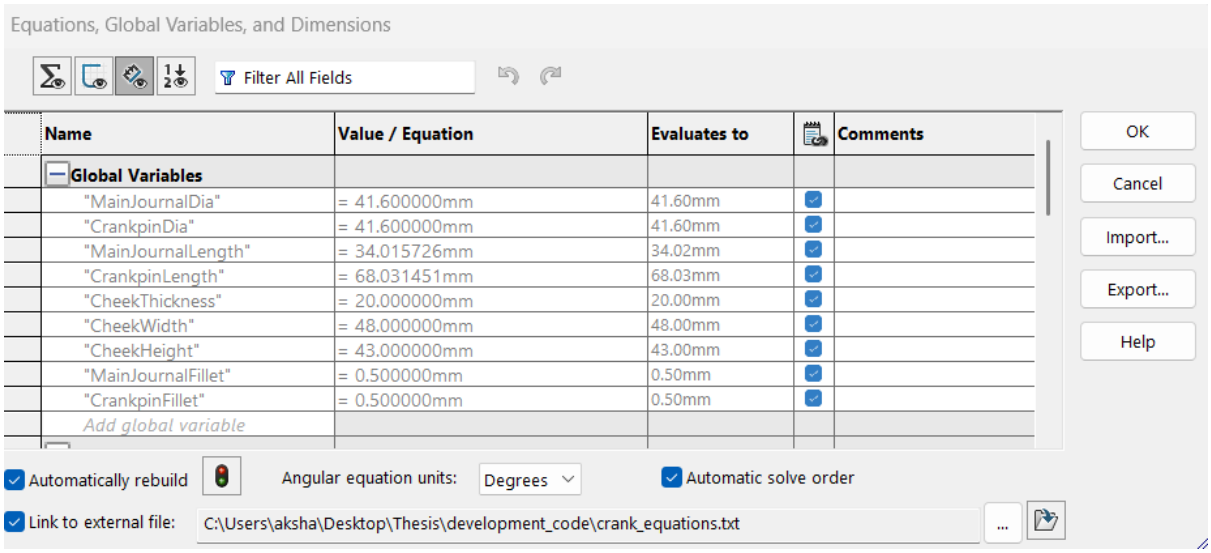

Figure 4.28: The SolidWorks Equation Manager, showing the global variables successfully imported from the MATLAB-generated file.

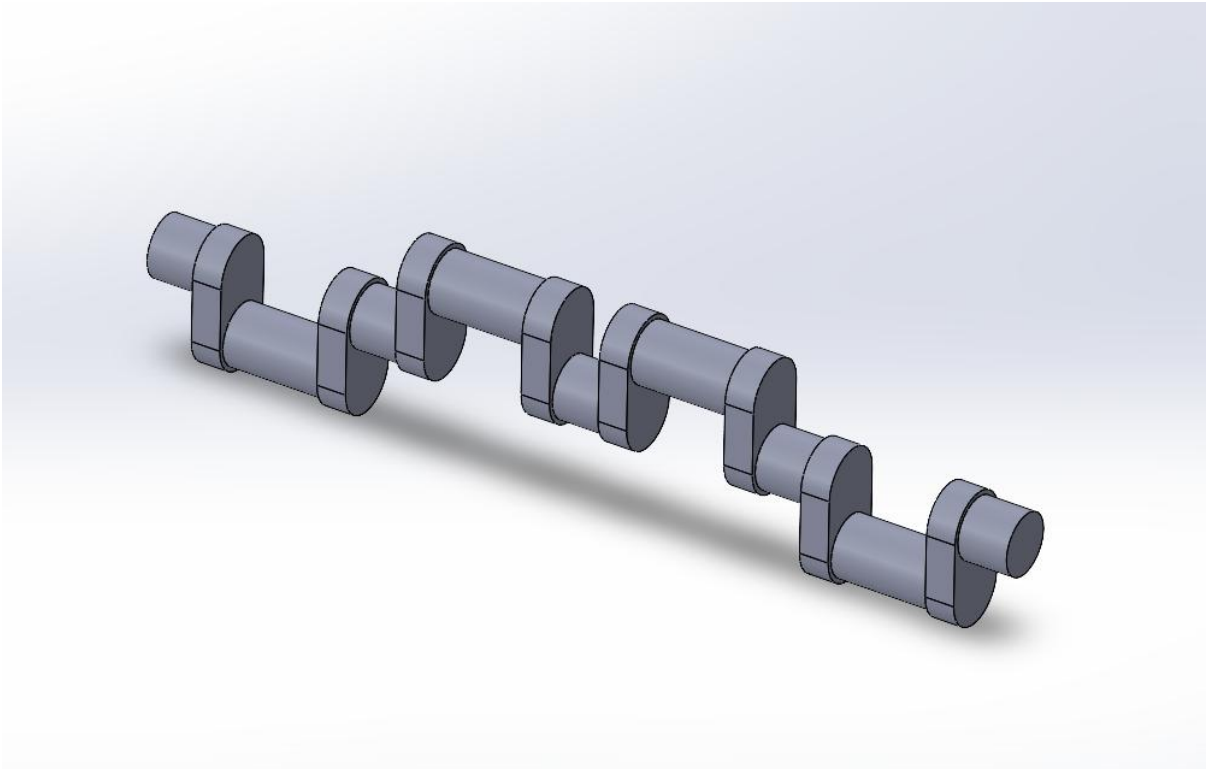

Figure 4.29: The automatically generated crankshaft solid model after the macro has completed the procedural generation.

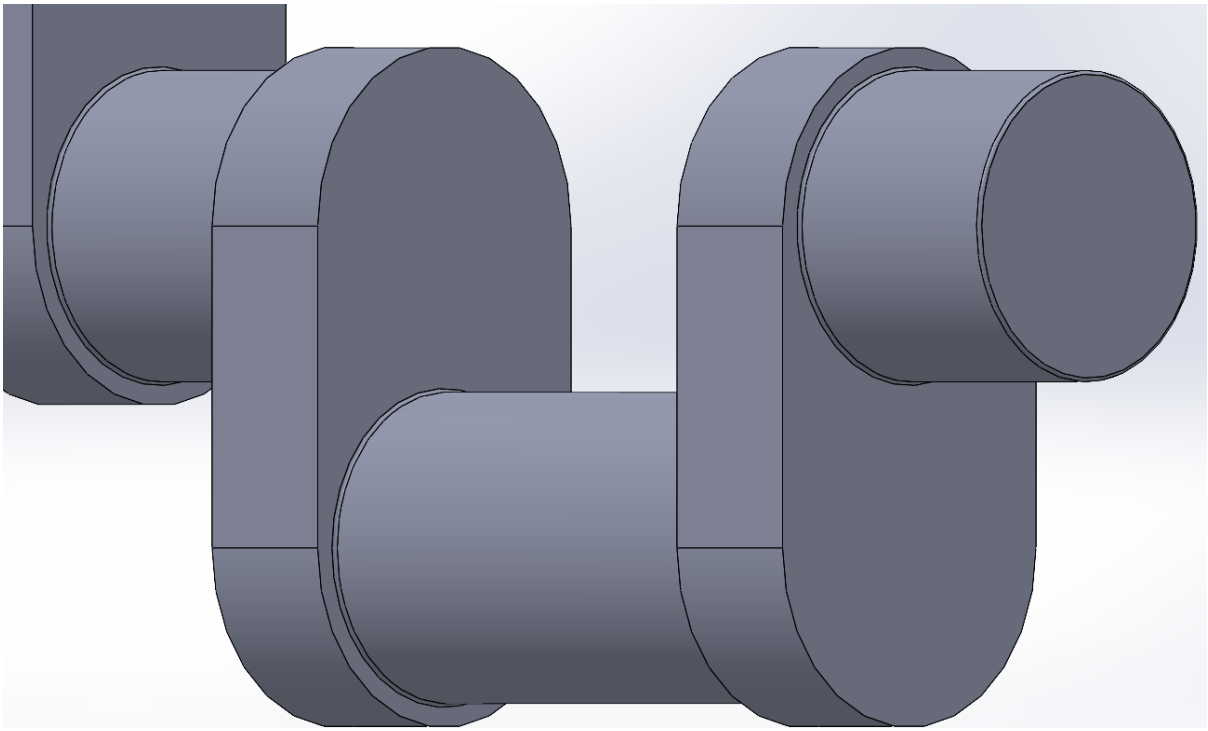

Figure 4.30: A detailed, close-up view of the final, completed 3D CAD model with manually applied fillets at a crankpin-web junction.

4.10. Chapter Summary and Validation

This chapter has presented a comprehensive case study designed to validate the end-to-end functionality of the proposed computational framework. By successfully navigating the entire workflow from initial parameter definition to final CAD model generation, the case study provides objective evidence that the framework meets its primary objectives. The key validation points demonstrated by the preceding results are summarized as follows:

- **Integrated and User-Guided Workflow:** The sequential progression through the various GUI modules confirms that the framework provides a logical, repeatable, and user-friendly process that effectively guides the engineer through the complex stages of crankshaft design.

- **Seamless Data Integration:** The case study demonstrates the successful and automatic transfer of data between analytical modules. The pressure curve from the thermodynamic analysis was correctly used to calculate forces, which in turn were used to calculate stresses and, ultimately, to drive the final geometry, proving the robust integration of the system.

- **Validation of the Iterative Design Loop:** The progression from a "NOK" status in the preliminary design to an "OK" status in the validation stage is a critical validation of the framework's core function as an interactive design tool. It proves the system's ability to identify a non-viable design and empower the user to rapidly iterate to a structurally sound solution.

- **Comprehensive Dynamic Analysis Capabilities:** The successful generation of the Campbell Diagram, multi-cylinder harmonic spectrum, and final forced vibration response plots validates the framework's capacity for advanced dynamic analysis. It demonstrates the ability to identify critical speeds and accurately predict the crankshaft's dynamic behaviour under operational load.

- **End-to-End Process Completion:** The final generation of a complete 3D CAD model serves as the ultimate proof of the framework's success. It confirms that the entire process, from abstract engineering requirements to a tangible, manufacturable design baseline, is fully realized within the integrated system.

In summary, the results presented in this chapter confirm that the computational framework is a capable and effective tool that successfully addresses the limitations of traditional, fragmented design workflows.

5. Conclusion

This thesis presents a novel computational framework for the automated design and generation of complex mechanical components, using the crankshaft as a comprehensive case study. The work transcends a simple automation script by integrating a robust engineering analysis backend with an iterative, user-driven validation process, thereby establishing a unified and data-centric design methodology. The primary objective was to validate a systematic workflow that streamlines the design cycle, minimizes manual intervention, and ensures the structural and dynamic integrity of the final component from the earliest conceptual stages.

The decision to focus on the crankshaft was a deliberate and foundational choice for this research. As the heart of an internal combustion engine, the crankshaft is not only a complex geometric form but also a component that embodies a confluence of critical engineering disciplines. It is subjected to a combination of extreme torsion and bending loads, operates under high-frequency dynamic forces, and is highly susceptible to destructive vibrations. Therefore, a successful computational framework for a crankshaft inherently requires the integration of kinematic analysis, structural mechanics, and advanced vibrational dynamics. By demonstrating the framework's capability to handle the multifaceted challenges presented by this single, vital component, we validate its potential for broader application to a wide range of other high-stress engine parts.

The developed system is founded on a rigorous computational core that addresses the multi-disciplinary requirements of crankshaft design. Initially, the framework performs a comprehensive kinematic analysis to model the precise motion of the piston and connecting rod. This analysis serves a dual purpose: it not only provides foundational data for subsequent dynamic calculations but also computes a set of preliminary dimensional parameters. This output initiates a crucial iterative dimensioning and validation phase. The user can then evaluate these initial dimensions and refine their inputs, with the system providing real-time feedback until the design achieves a satisfactory status. This interactive loop is a core innovation of the framework, as it ensures that the final design is not only computationally sound but also optimized by the user's engineering judgment and experience.

Upon user approval of the refined dimensions, the framework proceeds to its most significant contribution: the advanced dynamic and modal analysis. The system performs a forced

vibration analysis to evaluate the crankshaft's response to the periodic engine excitation forces. Concurrently, a modal analysis is conducted to compute the crankshaft's natural frequencies and corresponding mode shapes. The convergence of these analyses is visually represented by a Campbell Diagram, which serves as a critical tool for identifying potential resonance points and ensuring the design's operational stability by steering clear of destructive critical speeds. This integration of complex vibrational theory into an automated, iterative design tool represents a significant advancement over traditional, uncoupled design processes.

The successful implementation of this framework demonstrates a pathway to significant efficiency gains in the design process of mechanical components. By automating complex and computationally intensive tasks, the system effectively compresses a traditionally weeks-long design cycle into a matter of hours. This reduction in lead time not only accelerates the prototyping and development of new engine designs but also empowers engineers to explore a far greater range of design alternatives with a high degree of confidence. The modular architecture of the framework also provides a clear path for future expansion. Future research can focus on integrating advanced finite element analysis (FEA) for stress and strain evaluation, applying optimization algorithms to search for the most efficient design parameters, or extending the methodology to other critical components such as camshafts or connecting rods. This work thus lays the foundation for a new generation of intelligent, data-driven design tools.

The culmination of this process is the generation of a foundational Computer-Aided Design (CAD) model, which validates the framework's end-to-end capability. This is achieved through a robust, decoupled workflow where the MATLAB environment first exports the final, verified geometric parameters to a set of external text files. The user is then guided to the SolidWorks environment, where a custom macro reads these files and programmatically constructs the 3D solid model. This method creates a reliable "digital thread" from analysis to geometry, bridging the gap between theoretical computation and a tangible design asset ready for further refinement.

In summary, this thesis has successfully validated a comprehensive and integrated computational framework that addresses critical bottlenecks in the design of high-performance mechanical components. By seamlessly combining kinematic modelling with advanced dynamic and modal analysis, and by incorporating a user-driven iterative loop, this work moves

beyond traditional design methods. The resulting system is not merely an automation tool but a new paradigm for engineering design, offering a blueprint for creating more efficient, reliable, and sophisticated mechanical systems.

References

[1] Delprete, C., Tornincasa, S., Bonisoli, E., & Rosso, C. (2008). *A Tool for First Attempt Functional Drawing of Engine Main Components*. TCN CAE 2008 International Conference on Simulation Based Engineering and Sciences.

[2] Feese, T., & Hill, C. (2001). Guidelines for Improving Reliability of Reciprocating Machinery by Avoiding Torsional Vibration Problems. Gas Machinery Conference.

[3] Dubensky, R. G. (2002). *Crankshaft Concept Design Flowchart for Product Optimization* (SAE Technical Paper 2002-01-0770). SAE International.

[4] Rosso, C., Delprete, C., Bonisoli, E., & Tornincasa, S. (2011). *Integrated CAD/CAE Functional Design for Engine Components and Assembly* (SAE Technical Paper 2011-01-1071). SAE International.

[5] Delprete, C., Pregno, F., & Rosso, C. (2009). *Internal Combustion Engine Design: a Practical Computational Methodology* (SAE Technical Paper 2009-01-0477). SAE International.

[6] Kliebhan, F. (1966). *Fundamentals of Crankshaft Design* (SAE Technical Paper 660472). SAE International.

[7] Okamura, H., & Arai, S. (2001). Experimental Modal Analysis for Cylinder Block-Crankshaft Substructure Systems of Six-cylinder In-line Diesel Engines (SAE Technical Paper 2001-01-1421). SAE International.

[8] Honda, Y., Wakabayashi, K., Kodama, T., & Kihara, R. (2000). *A Basic Study on Reduction of Cylinder Block Vibrations for Small Diesel Cars* (SAE Technical Paper 2000-01-0527). SAE International.

[9] Spiteri, P., Ho, S., & Lee, Y.-L. (2007). Assessment of bending fatigue limit for crankshaft sections with inclusion of residual stresses. *International Journal of Fatigue*, 29(2), 318–329.

[10] Silva, F. S. (2003). Analysis of a vehicle crankshaft failure. *Engineering Failure Analysis*, 10(5), 605–616.

[11] Kang, Y., Sheen, G.-J., Tseng, M.-H., Tu, S.-H., & Chiang, H.-W. (1998). Modal analyses and experiments for engine crankshafts. *Journal of Sound and Vibration*, 214(3), 413–430.

[12] Heywood, J. B. (2018). Internal Combustion Engine Fundamentals (2nd ed.). McGraw-Hill Education.

[13] Taylor, C. F. (1985). The Internal-Combustion Engine in Theory and Practice, Vol. 1: Thermodynamics, Fluid Flow, Performance. MIT Press.

[14] Budynas, R. G., & Nisbett, J. K. (2020). Shigley's Mechanical Engineering Design (11th ed.). McGraw-Hill Education.

[15] Rao, S. S. (2017). Mechanical Vibrations (6th ed.). Pearson.

[16] Oberg, E., Jones, F. D., Horton, H. L., & Ryffel, H. H. (2012). *Machinery's Handbook* (29th ed.). Industrial Press.

Appendix A: Selected Source Code

This appendix contains excerpts from the key MATLAB scripts developed for this thesis, presented in the logical order of the framework's analytical workflow. The code is provided to demonstrate the implementation of the core engineering logic.

A.1: Thermodynamic Cycle Calculation

This function calculates the theoretical in-cylinder pressure curve based on user-defined thermodynamic parameters for an Otto cycle. This pressure curve is a fundamental input for all subsequent load and torque calculations. The code demonstrates the application of thermodynamic principles to generate the pressure-crank angle diagram.

```
function [crank_angle_deg_total, pressure_bar_total] = calculate_otto_cycle(T1, p1_bar, compression_ratio, p3_bar, k, D, r, l, n)
    % Calculates the pressure-crank angle curve for an ideal Otto cycle.
    %
    % Inputs:
    %   T1: Initial Temperature (K)
    %   p1_bar: Initial Pressure (bar)
    %   compression_ratio: Engine compression ratio
    %   p3_bar: Peak Combustion Pressure (bar)
    %   k: Specific heat ratio (gamma)
    %   D, r, l: Bore, crank radius, conrod length (m)
    %   n: Number of discretization points for the 720-degree cycle
    %
    % Outputs:
    %   crank_angle_deg_total: Vector of crank angles from 0 to 720 degrees
    %   pressure_bar_total: Corresponding pressure vector (bar)

    % --- 1. Initialize Parameters ---
    R_gas = 287; % Gas constant for air in J/(kg*K)
    p1_Pa = p1_bar * 1e5; % Convert input pressure to Pascals
    p3_Pa = p3_bar * 1e5;

    % --- 2. Calculate Characteristic Points of the Cycle ---
    % V1: Cylinder volume at Bottom Dead Center (BDC)
    V1 = compression_ratio / (compression_ratio - 1) * pi * (D^2) * r / 2;
    % V_clearance: Clearance volume at Top Dead Center (TDC)
    V_clearance = V1 / compression_ratio;

    rho1 = p1_Pa / (R_gas * T1); % Initial density
    mass_air = rho1 * V1;

    % Calculate pressure at other key points
    p2_Pa = p1_Pa * (compression_ratio)^k; % End of compression
    v_tdc = V_clearance / mass_air;      % Specific volume at TDC
    p4_Pa = p3_Pa * (1 / compression_ratio)^k; % End of expansion

    % --- 3. Generate Pressure Curve for Each Stroke ---
    num_points_per_stroke = n / 4;
    crank_angle_stroke = linspace(0, 180, num_points_per_stroke);
    theta_rad_stroke = deg2rad(crank_angle_stroke);

    % Piston displacement from TDC as a function of crank angle
    displacement = @(theta) r * (1 - cos(theta) + (l/r) * (1 - sqrt(1 - (r/l)^2 .* sin(theta).^2)));

    % --- Intake Stroke (0-180 deg) ---
    crank_angle_intake = crank_angle_stroke;
    pressure_intake = ones(1, num_points_per_stroke) * p1_bar;

    % --- Compression Stroke (180-360 deg) ---
    crank_angle_compression = crank_angle_stroke + 180;
```

```
    volume_compression = (displacement(deg2rad(crank_angle_compression - 180))) * (pi/4 * D^2) + V_clearance;
    specific_vol_comp = volume_compression / mass_air;
    pressure_compression = (p2_Pa * (v_tdc ./ specific_vol_comp).^k) / 1e5; % Convert back to bar

    % --- Expansion Stroke (360-540 deg) ---
    crank_angle_expansion = crank_angle_stroke + 360;
    volume_expansion = (displacement(deg2rad(crank_angle_expansion - 360))) * (pi/4 * D^2) + V_clearance;
    specific_vol_exp = volume_expansion / mass_air;
    pressure_expansion = (p3_Pa * (v_tdc ./ specific_vol_exp).^k) / 1e5; % Convert back to bar

    % --- Exhaust Stroke (540-720 deg) ---
    crank_angle_exhaust = crank_angle_stroke + 540;
    pressure_exhaust = ones(1, num_points_per_stroke) * p1_bar;

    % --- 4. Assemble and Save the Final Pressure Curve ---
    % The pressure vectors from each stroke are concatenated. A small transition
    % is added for the constant-volume combustion and exhaust blowdown phases.
    crank_angle_deg_total = [crank_angle_intake, crank_angle_compression, 360, crank_angle_expansion, 540,
crank_angle_exhaust];
    pressure_bar_total = [pressure_intake, fliplr(pressure_compression), p3_bar, pressure_expansion, p4_Pa/1e5,
fliplr(pressure_exhaust)];

    % Sort by crank angle to ensure correct order
    [crank_angle_deg_total, sort_idx] = sort(crank_angle_deg_total);
    pressure_bar_total = pressure_bar_total(sort_idx);

    save('pressione.mat', 'crank_angle_deg_total', 'pressure_bar_total');
end
```

A.2: Kinematic and Dynamic Load Analysis

This function is the core of the load analysis. It calculates the primary and secondary inertia forces for each cylinder, the resultant unbalanced forces and moments for the entire engine, and the total gas force from the pressure curve. It then combines these to determine the tangential and radial forces acting on each crankpin, which are subsequently used to calculate the engine torque.

```
function [Resultant_X1, Resultant_X2, Resultant_Y1, Resultant_Y2, Moment_Y1, Moment_Y2, Moment_X1, Moment_X2, ...
        Tangential_Force, Radial_Force, Total_Torque, Mean_Torque] = calculate_engine_loads(...)

    % This function calculates the resultant forces, moments, and torques for a multi-cylinder engine.
    %
    % Inputs:
    %   mM, mB, mPAS, mC: Component masses (kg)
    %   rC, omega, r, l: Geometric and operational parameters (SI units)
    %   n, N: Discretization points and number of cylinders
    %   phase_angles_deg, bank_angles_deg, cylinder_offsets: Engine configuration arrays
    %   pressure_curve, firing_order: Combustion data

    % --- 1. Initialize Parameters and Kinematic Relationships ---
    crank_angle_deg = linspace(0, 720, n);
    crank_angle_rad = deg2rad(crank_angle_deg)';

    omega_rad_s = omega * pi/30; % Convert engine speed from RPM to rad/s
    phase_angles_rad = deg2rad(phase_angles_deg);
    bank_angles_rad = deg2rad(bank_angles_deg);

    % --- 2. Calculate Inertia Force Coefficients ---
    % These coefficients are based on the masses and geometry of the moving components.
    coeff_a = (mB*r*l2/l + mM*l4 - mC*rC) * omega_rad_s^2;
    coeff_b = (mPAS*r + mB*r + mM*l4 - mC*rC) * omega_rad_s^2;
    coeff_c = (mPAS*r + mB*r*l1/l) * (r/l) * omega_rad_s^2;

    % --- 3. Calculate Inertia Forces for Each Cylinder ---
    for i = 1:N
```

```
        effective_theta = crank_angle_rad + phase_angles_rad(i);

        % Primary and secondary inertia forces (X and Y components) for main bearings
        Force_Inertia_X1(:,i) = (coeff_a*sin(effective_theta)).*cos(bank_angles_rad(i)) +
(coeff_b*cos(effective_theta)).*sin(bank_angles_rad(i));
        Force_Inertia_Y1(:,i) = -(coeff_a*sin(effective_theta)).*sin(bank_angles_rad(i)) +
(coeff_b*cos(effective_theta)).*cos(bank_angles_rad(i));
        Force_Inertia_X2(:,i) = (coeff_c*cos(2*effective_theta)).*sin(bank_angles_rad(i));
        Force_Inertia_Y2(:,i) = (coeff_c*cos(2*effective_theta)).*cos(bank_angles_rad(i));
    end

    % --- 4. Calculate Resultant Unbalanced Forces and Moments ---
    Resultant_X1 = sum(Force_Inertia_X1, 2);
    Resultant_Y1 = sum(Force_Inertia_Y1, 2);
    Resultant_X2 = sum(Force_Inertia_X2, 2);
    Resultant_Y2 = sum(Force_Inertia_Y2, 2);

    % (Calculation of resultant moments is also performed here, omitted for brevity)
    % ...

    % --- 5. Calculate Gas Forces and Total Resultant Forces on Crankpin ---
    % Shift the pressure curve for each cylinder according to the firing order
    for i = 1:N
        cylinder_num = firing_order(i);
        phase_shift_index = (i-1);
        pressure_phased(cylinder_num, :) = PhaseShift_PressureVector(N, phase_shift_index, pressure_curve);
    end

    % Decompose gas force into X and Y components
    Gas_Force_X = -sin(bank_angles_rad) .* pressure_phased * (pi * (D^2) / 4);
    Gas_Force_Y = -cos(bank_angles_rad) .* pressure_phased * (pi * (D^2) / 4);

    % (Calculation of inertia forces on the crankpin is also performed here, omitted for brevity)
    % ...

    % Total resultant force on each crankpin (Inertia + Gas)
    Total_Force_X = Inertia_Force_Crankpin_X + Gas_Force_X;
    Total_Force_Y = Inertia_Force_Crankpin_Y + Gas_Force_Y;
    Total_Resultant_Force = sqrt(Total_Force_X.^2 + Total_Force_Y.^2);

    % --- 6. Decompose Forces and Calculate Torque ---
    for i = 1:N
        effective_theta = crank_angle_rad + phase_angles_rad(i);
        effective_beta = bank_angles_rad(i);

        % Decompose the total resultant force into tangential and radial components
        Tangential_Force(i, :) = Total_Resultant_Force(i, :)' .* sin(effective_theta + effective_beta) ./ cos(effective_beta);
        Radial_Force(i, :)   = Total_Resultant_Force(i, :)' .* cos(effective_theta + effective_beta) ./ cos(effective_beta);
    end

    % Calculate torque for each cylinder
    Torque_Per_Cylinder = Tangential_Force * r;

    % Calculate total engine torque by summing individual cylinder torques according to firing order
    Total_Torque = zeros(1, n);
    for i = 1:N
        Total_Torque = Total_Torque + Torque_Per_Cylinder(firing_order(i), :);
    end

    Mean_Torque = sum(Total_Torque) / n;
end
```

A.3: Preliminary Dimensioning and Structural Verification

This section combines the logic from two key stages: the initial empirical dimensioning and the subsequent stress/fatigue verification. The first part of the function calculates a baseline set of geometric dimensions for the crankshaft based on empirical engineering formulas. The second part performs a rigorous stress analysis, calculating the combined bending and torsional

stresses at critical locations (crankpin, crank web, and main journal) and determines the minimum required journal diameter based on fatigue life criteria.

```
function [initial_dimensions, verification_results] = perform_structural_analysis(...)
    % Performs initial dimensioning and subsequent stress/fatigue verification.
    %
    % Inputs:
    %   max_pressure_Pa: Peak in-cylinder pressure (Pascals)
    %   max_tangential_force, max_radial_force: Peak forces from load analysis (N)
    %   bore_m, stroke_m: Engine bore and stroke (meters)
    %   material_properties: Struct containing ultimate strength, allowable stress, etc.
    %
    % Outputs:
    %   initial_dimensions: A struct with calculated dimensions (d, B, H, etc.) in meters.
    %   verification_results: A struct with calculated stresses (MPa) and pass/fail status.

    % --- 1. Preliminary Dimensioning (based on empirical formulas) ---

    % Convert inputs for empirical formulas which may use different units (e.g., cm)
    bore_cm = bore_m * 100;
    stroke_cm = stroke_m * 100;
    max_pressure_kg_cm2 = max_pressure_Pa / 98066.5;

    % Calculate initial crankpin diameter 'd' based on bore and pressure
    % (Example empirical formula)
    d_cm = 0.115 * k_factor * (bore_cm * (max_pressure_kg_cm2^2 * L3_cm^2 + A_factor * k1_factor * stroke_cm^2)^(1/2))^(1/3);

    % Calculate other dimensions as ratios of the crankpin diameter
    H_cm = 1.3 * d_cm; % Crank web height
    L1_cm = d_cm;      % Main journal length

    % Calculate crank web width 'B' based on stress limits
    B_cm = ((6*cc_cm + H_cm) * psi_factor * bore_cm^2 * max_pressure_kg_cm2 / (5 * H_cm^2 *
alternating_stress_limit_kg_cm2))^(1/3);

    % Convert final calculated dimensions back to SI units (meters)
    initial_dimensions.d = d_cm / 100;
    initial_dimensions.H = H_cm / 100;
    initial_dimensions.B = B_cm / 100;
    % ... and so on for all other dimensions ...

    % --- 2. Stress and Fatigue Verification ---

    % Use the calculated or user-defined dimensions (in meters) for verification
    d = initial_dimensions.d;
    B = initial_dimensions.B;
    H = initial_dimensions.H;
    % ... etc. ...

    allowable_stress_Pa = material_properties.allowable_stress;

    % --- Analysis at Crankpin ---
    bending_moment_1 = max_tangential_force * cylinder_distance / 4;
    bending_moment_2 = max_radial_force * cylinder_distance / 4;
    torsional_moment_1 = max_tangential_force * crank_radius / 2;

    section_modulus_bending = (pi / 32) * d^3;
    section_modulus_torsion = 2 * section_modulus_bending;

    bending_stress_1 = bending_moment_1 / section_modulus_bending;
    bending_stress_2 = bending_moment_2 / section_modulus_bending;
    total_bending_stress = sqrt(bending_stress_1^2 + bending_stress_2^2);
    shear_stress_1 = torsional_moment_1 / section_modulus_torsion;

    % Combined stress at crankpin using Von Mises criterion
    combined_stress_crankpin_Pa = sqrt(total_bending_stress^2 + 4 * shear_stress_1^2);
    verification_results.stress_crankpin_MPa = combined_stress_crankpin_Pa * 1e-6;

    % --- Analysis at Crank Web ---
    bending_moment_3 = max_radial_force * web_offset / 2;
    bending_moment_4 = max_tangential_force * crank_radius / 2;
```

```
    section_modulus_web_1 = (H * B^2) / 6;
    section_modulus_web_2 = (B * H^2) / 6;

    bending_stress_3 = bending_moment_3 / section_modulus_web_1;
    bending_stress_4 = bending_moment_4 / section_modulus_web_2;
    compressive_stress = max_radial_force / (2 * B * H);

    % Combined stress at crank web (superposition)
    combined_stress_web_Pa = bending_stress_3 + bending_stress_4 + compressive_stress;
    verification_results.stress_web_MPa = combined_stress_web_Pa * 1e-6;

    % (Analysis at Main Journal is similar and omitted for brevity)
    % ...

    % --- Fatigue Analysis: Calculate Minimum Required Diameter ---
    term1 = (max_tangential_force * cylinder_distance / 4)^2;
    term2 = (max_radial_force * cylinder_distance / 4)^2;
    term3 = 4 * (max_tangential_force * crank_radius / 4)^2;

    min_diameter_m = ((32 / (pi * allowable_stress_Pa)) * sqrt(term1 + term2 + term3))^(1/3);
    verification_results.min_diameter_mm = min_diameter_m * 1000;

    % --- Final Pass/Fail Status ---
    if verification_results.stress_crankpin_MPa < allowable_stress_Pa * 1e-6 && ...
       verification_results.stress_web_MPa < allowable_stress_Pa * 1e-6
        verification_results.status = "OK";
    else
        verification_results.status = "NOK";
    end
end
```

A.4: Dynamic and Vibrational Analysis

This section details the code for the torsional vibration analysis, which is divided into three main parts: modal analysis to find the system's inherent properties, harmonic analysis to define the excitation forces, and forced vibration analysis to calculate the final dynamic response.

A.4.1: Modal Analysis

This function is the core physics engine for the dynamic analysis. It constructs the global stiffness [K] and mass [J] matrices for the multi-degree-of-freedom lumped-mass model of the crankshaft. It then solves the generalized eigenvalue problem to find the system's torsional natural frequencies and corresponding mode shapes.

```
function [natural_frequencies, mode_shapes, K_matrix] = calculate_modal_properties(...)
    % Calculates the natural frequencies and mode shapes of the crankshaft system.
    %
    % Inputs:
    %   G: Shear Modulus (Pa)
    %   N: Number of cylinders
    %   Jpul, Jvol, Jd, Jo: Component inertias (kg*m^2)
    %   mB, mPAS: Component masses (kg)
    %   r, l, l1, l2: Geometric parameters (m)
    %   d_mm, B_mm, H_mm, L1_mm, L2_mm, Dvol_mm, Lvol_mm: Dimensions (mm)
    %
    % Outputs:
    %   natural_frequencies: Vector of natural frequencies (rad/s)
    %   mode_shapes: Matrix of mode shapes (column vectors)
    %   K_matrix: The global stiffness matrix (Nm/rad)
```

```
    % --- 1. Robust Unit Conversions ---
    % This block ensures all calculations are performed in base SI units (meters).
    % It checks if input dimensions appear to be in mm (value > 1) and converts them.
    if d_mm > 1, d_m = d_mm * 1e-3; else, d_m = d_mm; end
    if B_mm > 1, B_m = B_mm * 1e-3; else, B_m = B_mm; end
    % ... (other conversions for H_m, L1_m, L2_m, Dvol_m, Lvol_m) ...

    % Robustly convert masses, assuming they might be in grams
    if mB > 1, mB_kg = mB * 1e-3; else, mB_kg = mB; end
    if mPAS > 1, mPAS_kg = mPAS * 1e-3; else, mPAS_kg = mPAS; end

    % --- 2. Equivalent Moment of Inertia [J] ---
    mass_ratio_B_recip = mB_kg * l2 / l;
    mass_ratio_B_rot = mB_kg * l1 / l;
    lambda = r / l;

    J_equivalent_throw = Jd + mass_ratio_B_rot*r^2 + r^2*(mass_ratio_B_recip + mPAS_kg)*(8+2*lambda^2+lambda^4)/16 +
Jo*(4*lambda^2+lambda^4)/8;

    J_matrix = zeros(N+2, N+2);
    for i = 2:(N+1)
        J_matrix(i,i) = J_equivalent_throw;
    end
    J_matrix(1,1) = Jpul;
    J_matrix(N+2,N+2) = Jvol;

    % --- 3. Equivalent Torsional Stiffness [K] ---
    % Calculate equivalent length 'Lr' using a weighted average of empirical formulas
    % (Timoshenko, Ken Wilson, Zimanenko)
    % ... (calculation of Lr_1, Lr_3, Lr_4 omitted for brevity) ...
    Lr_weighted_avg = (1/10) * (2*Lr_3 + 3*Lr_4 + 4*Lr_3 + Lr_1);

    % Calculate equivalent stiffness of one crank throw section
    K_equivalent_throw = (G * pi * d_m^4) / (32 * Lr_weighted_avg);

    % Calculate stiffness of the flywheel shaft section
    Jp_flywheel = (pi * Dvol_m^4) / 32;
    K_flywheel_shaft = G * Jp_flywheel / Lvol_m;

    % Assemble the global stiffness matrix [K]
    K_matrix = zeros(N+2, N+2);
    % (Assembly logic omitted for brevity, constructs a tridiagonal matrix)
    % ...

    % --- 4. Solve Eigenvalue Problem ---
    % Solves [K]{phi} = omega^2 * [J]{phi}
    A = K_matrix * inv(J_matrix);
    [eigenvectors, eigenvalues] = eig(A);

    natural_frequencies = sqrt(abs(diag(eigenvalues)));
    mode_shapes = eigenvectors;
end
```

A.4.2: Multi-Cylinder Harmonic Analysis

This function calculates the total engine torque by summing the contributions from each cylinder according to the firing order. It then performs a Fourier analysis on this total torque signal to decompose it into its harmonic components (orders), which serve as the primary excitation forces for the vibration analysis.

```
function [harmonic_amplitudes, harmonic_orders] = calculate_multi_cylinder_harmonics(...)
    % Calculates the harmonic spectrum of the total multi-cylinder engine torque.
    % Inputs are loaded from .mat files.

    % --- 1. Reconstruct Total Engine Torque ---
    % This logic sums the individual cylinder torque vectors (Mt_vett)
```

```
    % based on the specified firing order (ordine_accensione).
    Total_Engine_Torque = zeros(1, num_points);
    for i = 1:N
        Total_Engine_Torque = Total_Engine_Torque + Torque_per_Cylinder(firing_order(i), :);
    end

    % --- 2. Perform Fourier Decomposition ---
    % The total engine torque is a periodic signal over 720 degrees.
    crank_angle_rad = deg2rad(crank_angle_deg_vector);

    max_order = 6.0;
    harmonic_orders = 0.5:0.5:max_order;
    harmonic_amplitudes = zeros(size(harmonic_orders));

    % Calculate Fourier coefficients for each harmonic order
    for i = 1:length(harmonic_orders)
        order = harmonic_orders(i);

        % For a 4-stroke engine, the k-th harmonic number for the Fourier series
        % corresponds to the (k/2)-th engine order.
        k_fourier = 2 * order;

        % Calculate coefficients Ak and Bk using numerical integration
        Ak = (2 / num_points) * sum(Total_Engine_Torque .* cos(k_fourier * crank_angle_rad));
        Bk = (2 / num_points) * sum(Total_Engine_Torque .* sin(k_fourier * crank_angle_rad));

        % Calculate the final amplitude for this order
        harmonic_amplitudes(i) = sqrt(Ak^2 + Bk^2);
    end
end
```

A.4.3: Forced Vibration Analysis

This function performs the final step of the dynamic analysis. It uses the principle of modal superposition to calculate the steady-state torsional vibration response of the crankshaft. It takes the system's modal properties (natural frequencies, mode shapes) and the multi-cylinder harmonic excitations as inputs and calculates the vibration amplitude at the flywheel and the dynamic torques in each shaft section across the full engine speed range.

```
function [flywheel_amplitudes, dynamic_torques] = calculate_forced_response(...)
    % Calculates the forced torsional vibration response using modal superposition.
    %
    % Inputs:
    %   natural_frequencies, mode_shapes, K_matrix: from modal analysis
    %   harmonic_amplitudes, harmonic_orders: from harmonic analysis
    %   engine_rpm_range: Vector of engine speeds to analyze
    %   damping_ratio: User-defined damping factor (zeta)

    % --- 1. Initialize and Pre-process ---
    % (Initialization of output arrays, conversion of RPM to rad/s, etc.)
    % ...

    % --- 2. Calculate Modal Parameters ---
    % For mass-normalized eigenvectors from MATLAB's eig(K,M), the generalized mass is 1.
    for k = 1:num_modes
        if abs(natural_frequencies(k)) > 1e-6 % Skip rigid-body mode
            generalized_masses(k) = 1;
        end
    end

    % --- 3. Main Calculation Loop ---
    % Iterate through each engine speed in the specified range
    for rpm_idx = 1:num_rpms

        complex_angular_displacements = zeros(num_masses, 1);
```

```
        % Iterate through each harmonic excitation order
        for harm_idx = 1:length(harmonic_orders)

            excitation_freq = harmonic_orders(harm_idx) * current_engine_omega;

            % Iterate through each natural mode of the system
            for k = 1:num_modes
                if generalized_masses(k) == 0, continue; end % Skip rigid-body mode

                phi_k = mode_shapes(:, k);
                omega_n_k = natural_frequencies(k);

                % --- Calculate Generalized Force for this mode ---
                % Distribute the total harmonic torque evenly across the N cylinders
                % and project onto the mode shape.
                torque_per_cylinder = harmonic_amplitudes(harm_idx) / N;
                generalized_force_k = 0;
                for cylinder_idx = 1:N
                    mass_index = cylinder_idx + 1;
                    generalized_force_k = generalized_force_k + (torque_per_cylinder * phi_k(mass_index));
                end

                % --- Calculate complex modal amplitude ---
                real_denom = (omega_n_k^2 - excitation_freq^2);
                imag_denom = (2 * damping_ratio * omega_n_k * excitation_freq);
                complex_denominator = (real_denom + 1i * imag_denom);

                modal_amplitude_complex = (generalized_force_k / generalized_masses(k)) / complex_denominator;

                % --- Sum the contribution of this mode to the total displacement ---
                complex_angular_displacements = complex_angular_displacements + (modal_amplitude_complex * phi_k);
            end
        end

        % Store the final vibration amplitude at the flywheel for this RPM
        flywheel_amplitudes(rpm_idx) = abs(complex_angular_displacements(end));

        % Calculate dynamic torques in each shaft section from relative displacements
        for s_idx = 1:num_sections
            relative_disp = complex_angular_displacements(s_idx + 1) - complex_angular_displacements(s_idx);
            dynamic_torques(s_idx, rpm_idx) = abs(section_stiffnesses(s_idx) * relative_disp);
        end
    end
end
```

A.5: Automated CAD Model Generation

This function represents the final step of the framework, where the verified analytical results are translated into a format suitable for generating a 3D model. The script gathers the final geometric dimensions, applies the necessary scaling to convert them to millimeters, and writes two separate text files. The first file is formatted for the SolidWorks Equation Manager to create a parametrically linked model, while the second file provides procedural instructions (like cylinder count and crank angles) for the custom automation macro.

```
function generate_cad_files(num_cylinders, crank_radius_m, crank_angles_deg, journal_dia, journal_length, crankpin_length, web_thickness, web_height)
    % Generates parameter files for automated SolidWorks CAD model creation.
    %
    % This script assumes that input dimensions like 'journal_dia' are stored
    % with a factor of 10 applied (e.g., 4.16 represents 41.6 mm). The
    % crank radius is assumed to be in meters.
```

```
    % --- 1. Map and Scale Variables to Millimeters (mm) ---
    main_journal_dia_mm    = journal_dia * 10;
    crankpin_dia_mm       = journal_dia * 10;
    main_journal_length_mm = journal_length * 10;
    crankpin_length_mm    = crankpin_length * 10;
    web_thickness_mm    = web_thickness * 10;
    web_height_mm        = web_height * 10;
    crank_throw_radius_mm       = crank_radius_m * 1000; % Convert meters to mm

    % Calculate the spacing for patterning the crank throws
    pattern_spacing_mm    = main_journal_length_mm + (web_thickness_mm * 2) + crankpin_length_mm;

    % --- 2. Write the Parametric Equations File for SolidWorks ---
    % This file links the MATLAB results to SolidWorks Global Variables.

    equations_filename = 'crank_equations.txt';
    fileID_eq = fopen(equations_filename, 'w');

    fprintf(fileID_eq, '"MainJournalDia" = %.6fmm\n',    main_journal_dia_mm);
    fprintf(fileID_eq, '"CrankpinDia" = %.6fmm\n',       crankpin_dia_mm);
    fprintf(fileID_eq, '"MainJournalLength" = %.6fmm\n', main_journal_length_mm);
    fprintf(fileID_eq, '"CrankpinLength" = %.6fmm\n',    crankpin_length_mm);
    fprintf(fileID_eq, '"WebThickness" = %.6fmm\n',    web_thickness_mm);
    fprintf(fileID_eq, '"WebHeight" = %.6fmm\n',       web_height_mm);
    fprintf(fileID_eq, '"CrankThrowRadius" = %.6fmm\n',       crank_throw_radius_mm);

    % Fillet radii are included for manual application in SolidWorks
    fprintf(fileID_eq, '"MainJournalFillet" = 2mm\n');
    fprintf(fileID_eq, '"CrankpinFillet" = 2mm\n');

    fclose(fileID_eq);

    % --- 3. Write the Procedural Configuration File for the VBA Macro ---
    % This file provides the instructions for the automation script.

    config_filename = 'crank_config.txt';
    fileID_cfg = fopen(config_filename, 'w');

    fprintf(fileID_cfg, 'NumCylinders:%d\n',      num_cylinders);
    fprintf(fileID_cfg, 'PatternSpacing:%.6f\n',    pattern_spacing_mm);

    % Format the crank angles as a comma-separated string
    angles_str = strjoin(string(crank_angles_deg), ',');
    fprintf(fileID_cfg, 'CrankAngles:%s\n', angles_str);

    fclose(fileID_cfg);

    disp('Successfully created crank_equations.txt and crank_config.txt');
end
```