UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Projection-based hyper-reduced order modeling of stress and reaction fields, and application of static condensation for multibody problems.

Sebastian Ares de Parga Regalado

Master in Numerical Methods in Engineering
Universitat Politècnica de Catalunya

Supervisors: Dr. Riccardo Rossi and M.Sc. Raul Bravo.

18th June, 2021

Abstract

Computational Mechanics' problems are often solved using the *Finite Element Method* (FEM). The resulting systems of equations may lead to large data and therefore, the solution requires high memory and time to be computed. This situation can be surpassed by applying *Reduced Order Modeling* (ROM) techniques, allowing the user to capture the system's dominant effects to build a high-fidelity reduced model that gives the possibility to predict and analyze the behaviour of a complex model using low computational resources within a micro time-step.

This paper aims to enrich the already implemented Kratos' *Rom Application* with a reconstruction of the *reaction* and *2nd Piola Kirkchhoff stress* fields. The applied methodology is a projection-based strategy using the *Proper Orthogonal Decomposition* together with a *Gappy Data* reconstruction technique. The gappy data comes from building a *hyper-reduced order model* (HROM).

A *surrogate model application* using *static condensation* and HROM techniques is proposed to show the possibility of solving *multibody systems* interfacing Kratos' ROM framework with *Mathworks* control capabilities in a fast and accurate way. The validation of the applied methodology is given by 3D complex models.

Acknowledgements

First of all, I would like to thank **Dr. Riccardo Rossi** for all the support and trust that he put in me during this year of work, who offered me the opportunity to work with him and his team in the development of surrogate and reduced-order models. This stage not only helped me to orient my goals and tastes for engineering and programming, but it also captivated me to move forward in the field of research and practical-theoretical production of scientific material through a Ph.D. in Civil Engineering with him as the supervisor. It is an honor to have developed material for the computational mechanics' software "Kratos Multiphysics", of which Riccardo is co-founder.

Within Kratos Multiphysics' research group, I want to thank the doctoral student **M.Sc. Raul Bravo**, who always offered me his help throughout the process of my research, to the extent that he was a decisive factor in changing the main topic of my thesis by allowing me to go deeply into his work alongside him in the implementation of model order reduction techniques for Kratos, reason for which I naturally registered him as a thesis supervisor. I extend huge gratitude to **Dr. Joaquín Hernandez** who always supported us with his knowledge when facing doubts and comments that arose during my internship and master's thesis.

I am very grateful to the Mathworks team, especially **Dr. Andreas Apostolatos** and **M.Sc. Steve Miller**, who always showed great interest in my work and answered all my doubts. Without this support, it would not have been possible to finalize the application of a surrogate model between Mathworks and Kratos.

I am grateful for the support of the **National Council of Science and Technology** (CONACYT) who, through a grant for living expenses and tuition, allowed me to do this postgraduate course at the UPC. I also extend my gratitude to the **Fundació Catalunya La Pedrera** for supplementing my living expenses with a scholarship that allowed me to enjoy the city and its surroundings, removing the economic concern that life as a student in a foreign country entails. Without these supports, my dream of studying abroad would not have been possible.

I would like to thank the **Universitat Politècnica de Catalunya** and the **Centre Internacional de Mètodes Numèrics a l'Enginyeria** that allowed me to enjoy academic and cultural exchange through their classes, events, seminars, projects, among many other academic activities. Extending this gratitude to its administrative staff, and especially to the secretary **Lelia Zielonka** who always attended and resolved my academic-administrative situations.

I show gratitude to my alma mater, the **National Autonomous University of Mexico** (UNAM), which gave me the academic and cultural resources necessary to expand my possibilities of making my dreams come true.

Without all the academic and personal support of previously mentioned individuals and institutions, this dream would not have come to fruition the way it did.

Dedication

Quisiera dedicar este trabajo especialmente a mi familia, quienes siempre me brindaron apoyo incondicional durante toda mi vida, buscando siempre mi felicidad bajo cualquier circunstancia.

A mi padre **Gonzalo Ares de Parga Álvarez**, quién siempre ha sido un ejemplo a seguir no solo por su trayectoria académica como profesor en física, sino por la persona que es en su día a día. Quién a través de la confianza, impulsa mis ganas de conseguir mis metas y objetivos, pero sobre todo la felicidad.

A mi madre **María de los Ángeles Regalado Castillo**, mi gran fuente de inspiración, bondad y alegría, brindándome siempre un apapacho inigualable el cual ha permitido que realice mis sueños con tranquilidad y comodidad. Su amor y preocupación por mi bienestar siempre acobija mí día a día y me hace ser siempre mejor persona que ayer.

A mi hermana **Ángela Ares de Parga Regalado**, a quién admiro por su destreza que se ve reflejada en su gran carrera académica como profesora de física en la universidad más prestigiosa de México. Agradezco mucho el apoyo incondicional que siempre me brindó tanto para situaciones académicas como para consejos de vida.

A mi hermano **Gonzalo Ares de Parga Regalado**, de quién tome grandes consejos de vida, y aquel que nunca deja de sorprenderme por sus logros académicos, profesionales y personales. El hermano que me enseñó que tener un objetivo de vida no es más que una estrategia de búsqueda de la felicidad.

A dos seres muy importantes, **Fifa y Gol**, quienes brindaron su amor y compañía incondicional durante toda mi etapa académica y gran parte de mi vida, y que siempre llenan mi vida de anécdotas y felicidad.

A mi novia **Simonetta Onofrietti Magrassi**, quien además de llenar esta aventura de innumerables anécdotas, alegría, amor y confianza, abrió las puertas de su casa recién iniciada esta etapa tan importante para mí.

A mis **amigos/as y compañeros/as**, a esa familia que uno va escogiendo durante la vida, a todos esos personajes que brindaron alegría, conocimiento, cultura, felicidad y confianza.

Dedico este trabajo a todo aquel que haya impactado mi vida de alguna manera, y que ha permitido que este sueño se haga realidad. Gracias.

Contents

List of Figures

List of Tables

PART I

INTRODUCTION

As the use of computational support tools has grown, Computational Mechanics, the subdiscipline of mechanics that develops and applies numerical methods with digital computer capabilities, has been developed immensely to find the solution to Multiphysics Engineering problems, with the main objective of understanding and harnessing the resources of nature. This great development coming from the improvement of computers and the availability of new calculation methods has allowed to analyze high-dimensional problems such as multi-scale, multibody, and control problems. However, today's technological opportunities demand to take advantage of techniques that allow performing analyzes with precision and quickness. One effective way to overpass this is to perform the so-called dimensionality reduction, which will enable to transform data from a high dimensional space into a low dimensional space retaining the most meaningful characteristics, properties, or information from the original set of data.

In structural engineering, the most common results of the analysis are the displacements, reactions, and stresses to which the structure under analysis is subjected. Obtaining a Reduced Order Model (ROM) will allow the user to modify input parameters to analyze the response (displacements, reactions, and stresses) of the structure to these modifications quickly and with error tolerances predefined by the user.

Many of the software that today allow the analysis of control systems or multi-body problems, only describe the global behavior of the system, because a full analysis of the results requires large amounts of memory and time. Arising the need to obtain a ROM that enables quick analysis and post-processing of the structural elements.

1 Objectives

The main objective is to make use of the capabilities of Kratos Multiphysics, an extremely powerful and efficient open-source software for solving a wide variety of mechanical problems, taking the already implemented base of Reduced Order Models (ROM) application that currently enables the post-processing of the displacements in structural problems and implement the necessary processes that will allow the post-processing of reactions and stresses. The implementations in Kratos Multiphysics are expected to allow interface with other computational software such as Simscape and Simulink (Mathworks). Mathworks, offers a very efficient user-friendly interface for modeling control systems and multibody problems, however, it does not allow going into detail in the physical responses of the structural components within this model (post-processing of a structural member), making a simplified analysis with low-fidelity results coming from the structure.
Kratos Multiphysics aims to provide Mathworks with a ROM that is sufficiently precise (up to user's input requirements) to increase the fidelity of its results, and in addition, allow detailed post-processing of results of the structures involved in the control or multi-body problem.

PART II

THEORETICAL BACKGROUND

2 Singular Value Decomposition

The Singular Value Decomposition (SVD) is a very reliable matrix factorization, providing a matrix decomposition that results into a low rank matrix approximation to be used for a wide variety of cases, in addition, it helps to find the best fit solution for a singular, under-determined, and/or over-determined linear system of equations of the form:

$$\mathbf{Ax} = \mathbf{b} \tag{1}$$

The SVD is also a basis algorithm of the *principal component analysis* (PCA), which as it's name indicates, focuses on finding the principal descriptive data for high-dimensional sets with a low-dimensional decomposition.

In computational mechanics, the solution of complex problems leads to large data sets, commonly saved as matrices. For instance, if the results of a series of experiments or simulations of *state-dimensions* n (e.g. the amount of degrees of freedom in a structural mechanics problem) are stored in a high dimensional matrix of snapshots, these snapshots represent low rank systems, meaning that only few patterns (dominant) explain or capture the high dimensional data [1].

2.1 Definition

Given a matrix data set $\mathbf{X} \in \mathbb{R}^{n\times m}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{x}_1 & \mathbf{x}_2 & \dots & \mathbf{x}_m \\ | & | & & | \end{bmatrix} \tag{2}$$

Where $\mathbf{x}_k \in \mathbb{R}^n$ is the snapshot vector of the simulation or experimental data. For instance, a typical application would be in image compression, where the snapshot vectors will be a column vector with all the pixels that an image contains, whereas for a structural mechanics problem it could be the displacements of a time-series simulation. This means that the state-dimension n (rows) will often refer to very high dimensional data. Meanwhile the columns "m" are called the snapshots of the matrix data set X. Usually, for computational mechanics simulations, $\mathbf{X}$ will have $n >> m$, and will be referred as the *Snapshot Matrix* throughout this paper.

The SVD is a unique matrix factorization of a real or complex matrix $\mathbf{X} \in \mathbb{R}^{n\times m}$ (for the purpose of this paper, consider only real matrices):

$$\mathbf{X} = \mathbf{U\Sigma V^T} \tag{3}$$

where $\mathbf{U}$ and $\mathbf{V}$ are $n \times n$ and $m \times m$ real unitary matrices respectively with orthonormal columns, and $\mathbf{\Sigma}$ is a $n \times m$ matrix with real, non-negative entries on the diagonal and zeros off

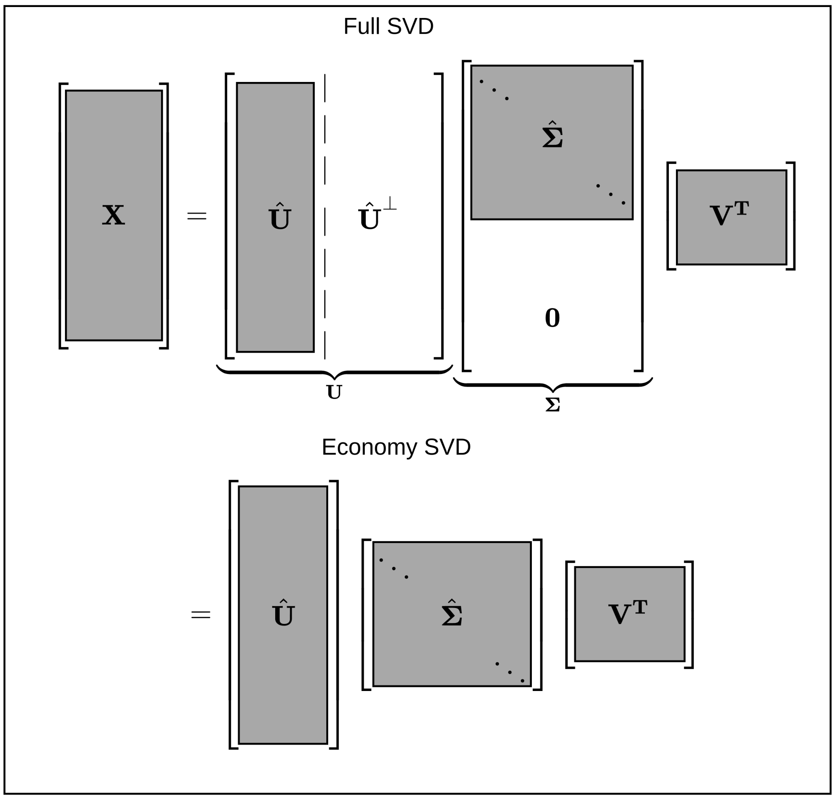


Figure 1: Full and economy SVD (Redrawn from [1])

the diagonal ordered from largest to smallest. From now on, $\mathbf{U}$, $\mathbf{V}$, and $\mathbf{\Sigma}$ may be referred as the matrices of left singular vectors, right singular vectors, and singular vectors respectively [1].

2.2 Economy SVD

When $n \geq m$, the singular vectors matrix $\mathbf{\Sigma}$ has at most m non-zero elements on the diagonal. Therefore, representing the singular values as a column vector:

$$\mathbf{\Sigma} = \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \tag{4}$$

Yielding the *economy* SVD:

$$\mathbf{X} = \mathbf{U\Sigma V^T} = \begin{bmatrix} \hat{\mathbf{U}} & \hat{\mathbf{U}}^{\perp} \end{bmatrix} \begin{bmatrix} \hat{\mathbf{\Sigma}} \\ \mathbf{0} \end{bmatrix} \mathbf{V^T} = \hat{\mathbf{U}}\hat{\mathbf{\Sigma}}\mathbf{V^T} \tag{5}$$

Where the columns of $\hat{\mathbf{U}}^{\perp}$ span a vector space that is complementary and orthogonal to that spanned by $\hat{\mathbf{U}}$. The rank of $\mathbf{X}$ is given by the number of the non-zero singular values [1]. To best describe the previously mentioned, please refer to the figure 1.

2.3 Truncated SVD

In [4], it is established that “The optimal rank-r approximation to $\mathbf{X}$, in a least-squares sense, is given by the rank-r SVD truncation $\tilde{\mathbf{X}}$”.

$$\tilde{\mathbf{X}} = \underset{\tilde{X},\ \text{s.t.}\ rank(\tilde{X})=r}{argmin} \|\mathbf{X} - \tilde{\mathbf{X}}\|_F = \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}} \tag{6}$$

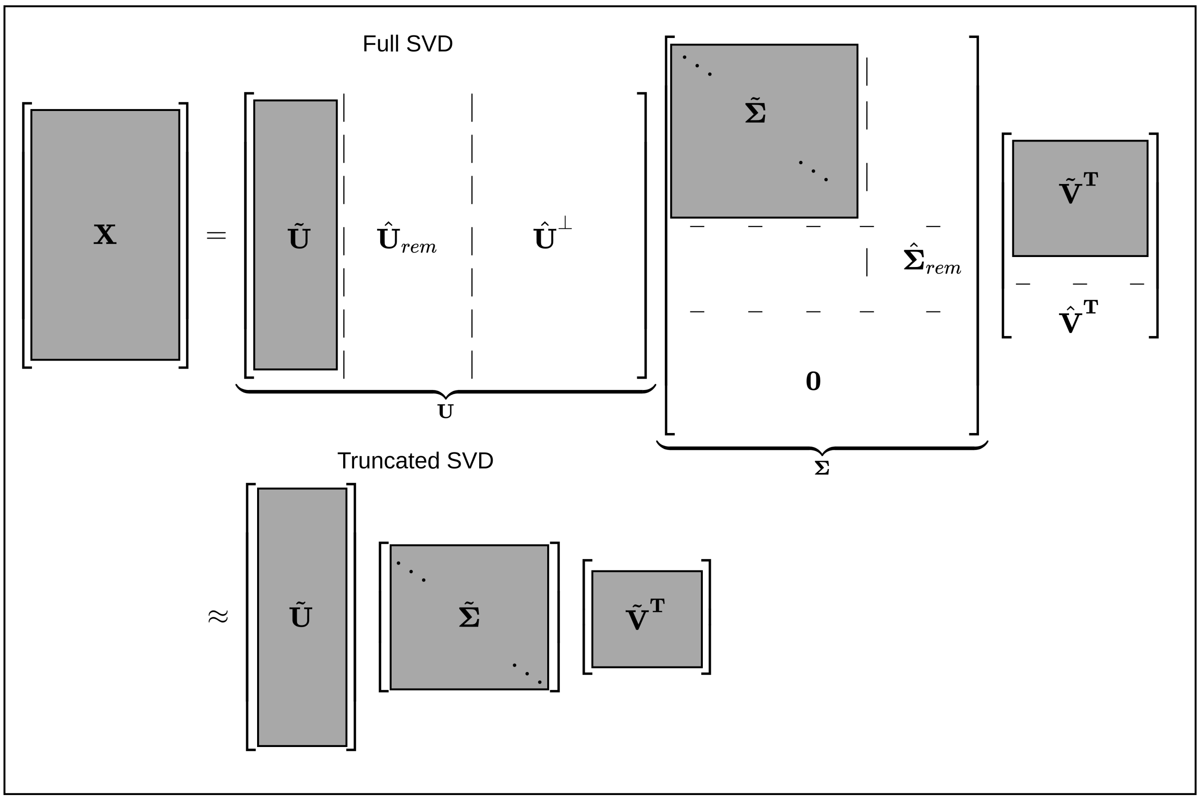

Full SVD
$\mathbf{X}$
$=$
$\tilde{\mathbf{U}}$
$\hat{\mathbf{U}}_{rem}$
$\hat{\mathbf{U}}^{\perp}$
$\mathbf{U}$
$\tilde{\mathbf{\Sigma}}$
$\hat{\mathbf{\Sigma}}_{rem}$
$\mathbf{0}$
$\mathbf{\Sigma}$
$\tilde{\mathbf{V}}^{\mathbf{T}}$
$\hat{\mathbf{V}}^{\mathbf{T}}$
Truncated SVD
$\approx$
$\tilde{\mathbf{U}}$
$\tilde{\mathbf{\Sigma}}$
$\tilde{\mathbf{V}}^{\mathbf{T}}$

Figure 2: Truncated SVD (Redrawn from [1])

Where $\tilde{\mathbf{U}}, \tilde{\mathbf{V}}$, and $\tilde{\mathbf{\Sigma}}$ contains the truncated most important information, and $||||_F$ denotes the *Frobenius norm*. Therefore, the SVD truncated for r singular values, approximates $\mathbf{X}$:

$$\mathbf{X} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}} \tag{7}$$

In figure 2, the truncated SVD is depicted.

2.4 Randomized SVD

To improve the computational cost of the decomposition of large matrices, the *randomized* numerical linear algebra is applied to obtain an accurate and efficient decomposition. Using random test matrices $\mathbf{P}$ to sample the column space and find a matrix $\mathbf{Q}$ with orthonormal columns to approximate the column space of a given matrix $\mathbf{X}$, such that:

$$||\mathbf{X} - \mathbf{Q}\mathbf{Q}^{\mathbf{T}}\mathbf{X}||_F^2 < \epsilon \tag{8}$$

This methods were first developed to improve the multiplication of high-dimensional matrices, and later expanded to the SVD theory. To explain the idea of randomized SVD, let us focus in the randomized SVD algorithm of Halko, Martinsson, and Tropp [5].

- The main and first step is to build a random test matrix $\mathbf{P} \in \mathbb{R}^{m \times r}$ to sample the column space of $\mathbf{X} \in \mathbb{R}^{n \times m}$:

  $$\mathbf{Z} = \mathbf{X}\mathbf{P} \tag{9}$$

  Construct a matrix $\mathbf{Q}$ whose columns form an orthonormal basis for the range of $\mathbf{Z}$ by computing the low-rank *QR decomposition*:

  $$\mathbf{Z} = \mathbf{Q}\mathbf{R} \tag{10}$$

- It is now possible to project $\mathbf{X}$ into a much smaller space with the help of $\mathbf{Q}$:

$$\mathbf{Y} = \mathbf{Q^T X} \tag{11}$$

To the extend that whenever the singular values decay rapidly, a fair approximation of $\mathbf{X}$ will be obtained:

$$\mathbf{X} \approx \mathbf{QY} \tag{12}$$

Recalling equation 11, obtain the SVD on Y:

$$\mathbf{Y} = \mathbf{U_Y \Sigma_Y V_Y^T} \tag{13}$$

Where, since $\mathbf{Q}$ is orthonormal and approximates the column space of $\mathbf{X}$:

$$\mathbf{\Sigma} = \mathbf{\Sigma_Y} \qquad \mathbf{V^T} = \mathbf{V_Y^T} \tag{14}$$

Therefore, substituting 14 into 13:

$$\mathbf{Y} = \mathbf{U_Y \Sigma V^T} \tag{15}$$

- Reconstruct the left singular vector of $\mathbf{X}$ with the use of $\mathbf{U_Y}$ and $\mathbf{Q}$:

$$\mathbf{U} = \mathbf{QU_Y} \tag{16}$$

Based on [1] section 1.8.

2.5 Image Processing

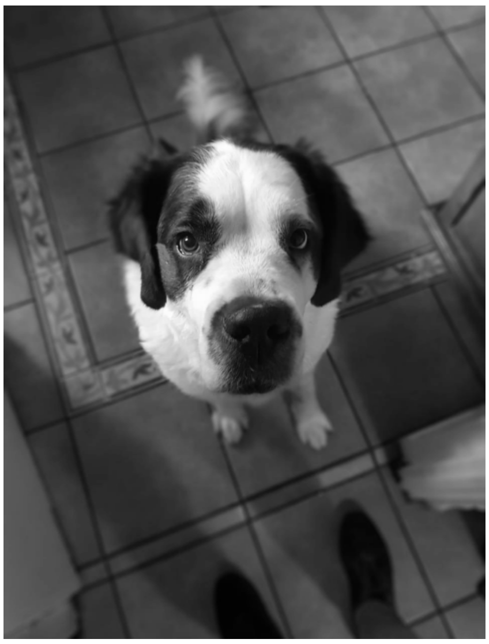
To show the idea of the previous sections and the SVD itself, it is provided a typical example of SVD application, *image processing.* A gray-scale image can play the role of a matrix $\mathbf{X} \in \mathbb{R}^{n \times m}$, where n and m represent the number of vertical and horizontal pixels of the image respectively. To show this practical example, the *numpy's* SVD library is going to be used truncating the SVD rank r to different values to capture the behavior of the approximation of the images. The sample image is a 1280x960 gray-scale image of "Fifa" (San Bernard dog) and was reconstructed with 5, 20, and 100 truncation values r. In figure 3, it can be seen that for a rank 5 reconstruction with less than 1% of storage, it can definitely be recognized the figure of a dog, meanwhile for a rank 20 reconstruction you can completely identify the specific dog's characteristics with less than 4% of the information, and finally, for a rank 100 you can even identify small details such as the shine on her eye. In figure 4 it can be noticed a fast decay on the singular values, and after around 100 modes it flattens out. In figure 5, it can be observed how singular values account for more than 50%, 70%, and 90% of the image variance for 5, 20, and 100 modes respectively.

2.6 Pseudo-Inverse

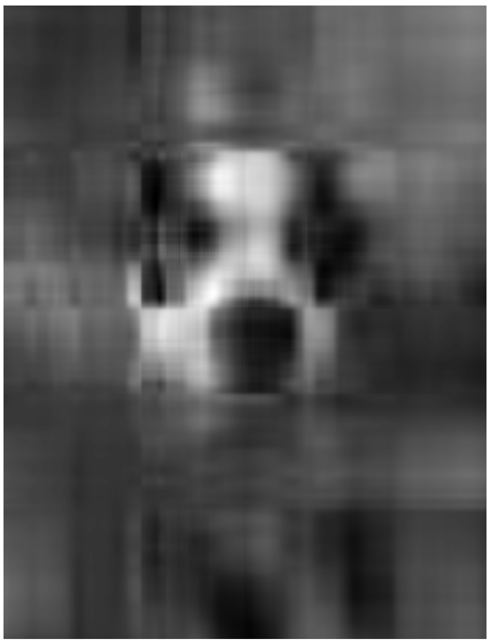
Physical and statistical problems many times lead to linear problems of the form:

$$\mathbf{Ax} = \mathbf{b} \tag{17}$$

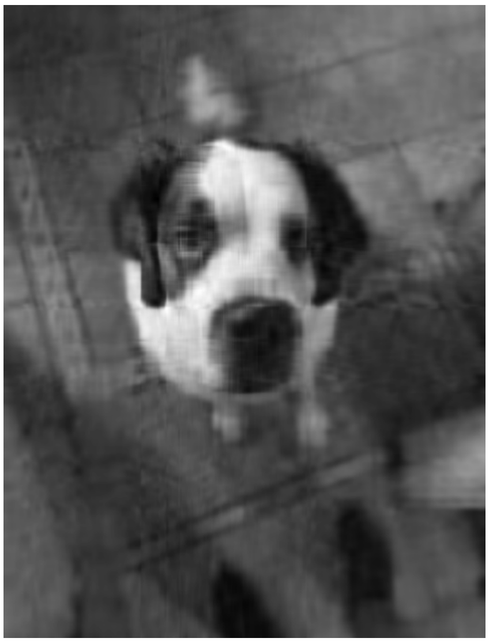
Where the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, $\mathbf{x} \in \mathbb{R}^{n \times m}$, and $\mathbf{b} \in \mathbb{R}^{n \times m}$, whose solution comes from the knowledge of the $\mathbf{A}$ and $\mathbf{b}$ matrices. A unique solution exists if the matrix $\mathbf{A}$ is invertible.



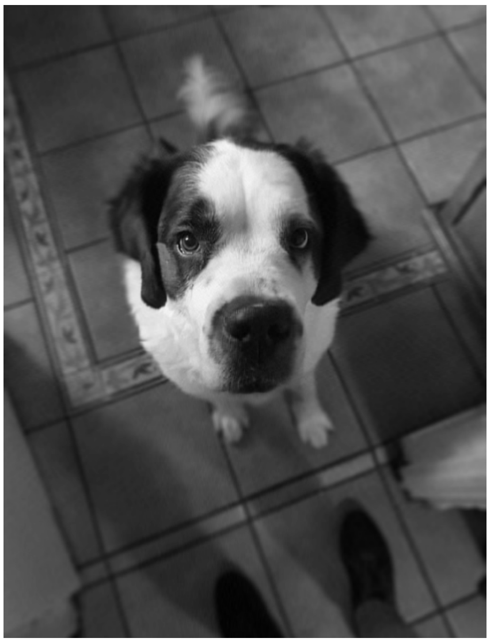
(a) Original



(b) $r = 5$, Storage percentage=0.911%



(c) $r = 20$, Storage percentage=3.646%



(d) $r = 100$, Storage percentage=18.229%

Figure 3: Image compression of Fifa with different truncation values r.

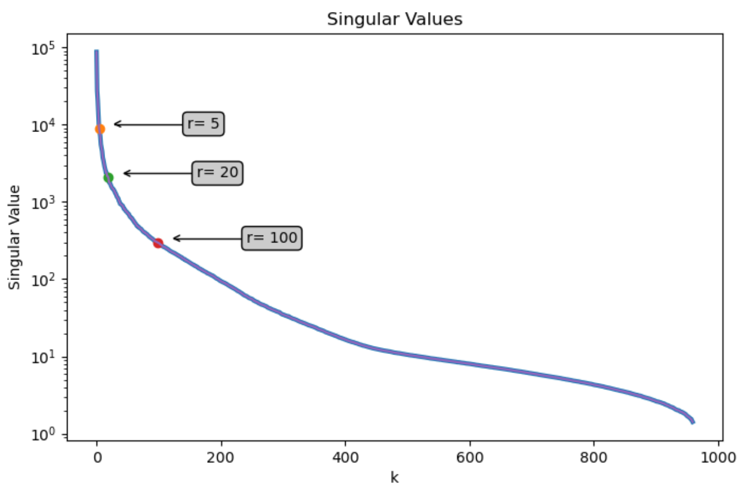


Figure 4: Singular Values

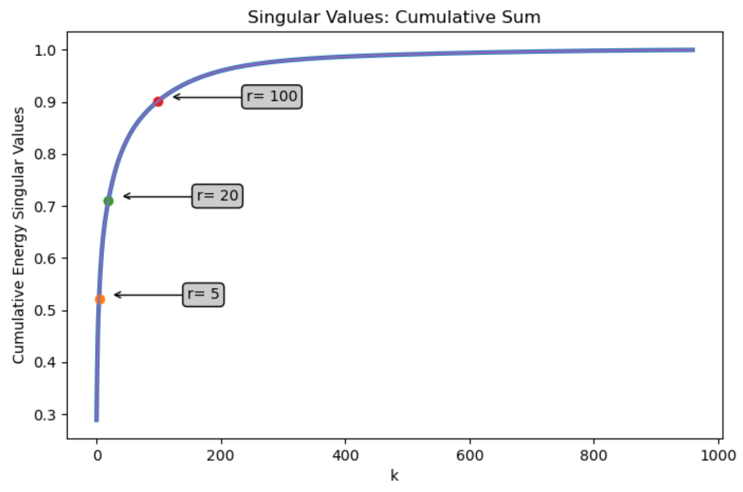


Figure 5: Cumulative Energy Singular Values

This solution is achieved by pre-multiplying 17 with the inverse of **A**:

$$\underbrace{\mathbf{A^{-1}A}}_{\mathbf{I}}\mathbf{x} = \mathbf{A^{-1}b} \tag{18}$$

$$\mathbf{x} = \mathbf{A^{-1}b} \tag{19}$$

Where **I** denotes the identity matrix.

However, many times the matrix **A** is either singular or rectangular, leading to a non-certain amount of solutions that could vary from none, one or infinite solutions. This paper will mainly focus in the rectangular case, where $\mathbf{A} \in \mathbb{R}^{n \times m}$ (usually $n >> m$, meaning that there are more equations than unknowns, this system is called *over-determined system*). One solution space for 17 is determined by the 4 fundamental spaces, and the SVD is the chosen technique to solve the minimization problem:

$$||\mathbf{A\tilde{x}} - \mathbf{b}||_2^2 \tag{20}$$

The truncated SVD of $\mathbf{A}$ yields:

$$\mathbf{A} \approx \tilde{\mathbf{U}}\tilde{\mathbf{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}} \tag{21}$$

And considering the *Moore-Penrose pseudo-inverse* [6]:

$$\mathbf{A}^{\dagger} = (\mathbf{A}^{\mathbf{T}}\mathbf{A})^{-1}\mathbf{A}^{\mathbf{T}} \tag{22}$$

For a matrix $\mathbf{A}$ of the form 21, the *left pseudo-inverse* [7] is defined as:

$$\mathbf{A}^{\dagger} = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^{\mathbf{T}} \tag{23}$$

Where:

$$\mathbf{A}^{\dagger}\mathbf{A} = \mathbf{I} \tag{24}$$

An example of a simple pseudo-inverse application is presented below.

2.7 Simple 1D example

To show a one dimensional linear approximation, a data that originally forms a "line" is proposed with some added noise to then reproduce a line that best fits the noisy data and compare it with the "original line". The over-determined system is of the form:

$$\mathbf{a}x = \mathbf{b} \tag{25}$$

$$x = \mathbf{a}^{\dagger}\mathbf{b} \tag{26}$$

Solving for x:

$$x = \tilde{\mathbf{V}}\tilde{\mathbf{\Sigma}}^{-1}\tilde{\mathbf{U}}^{\mathbf{T}}\mathbf{b} \tag{27}$$

In figure 6 it can be observed that it well approximates the true line regardless of the noisy data.

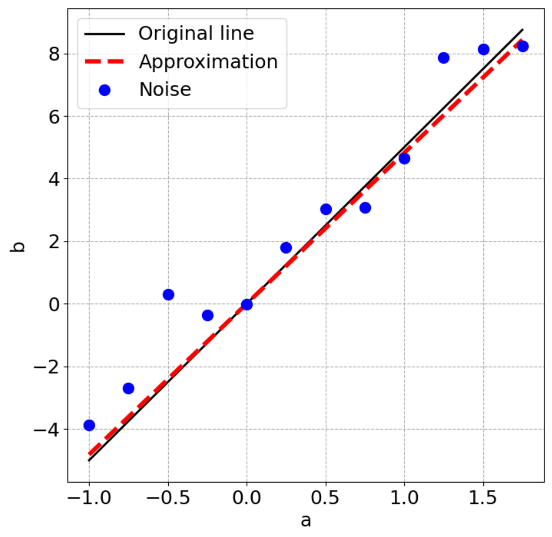


Figure 6: Left pseudo-inverse approximation for a one-dimensional over-determined system

3 Finite Element Method

The finite element method (FEM) is a numerical procedure for the solution of the equations that govern the behaviour of physical phenomena. These equations are of the form of differential equations. This method allows to find multiple solutions of the different spatial or time parameters of the phenomena by solving a numerical system.
In structural mechanics, FEM is a powerful procedure to compute the displacements, reactions, stresses, strains, among many other different parameters that may arise depending on the physical phenomena under study.
The basic idea is to decompose a continuum spatial medium into a *finite* set of small portions. The original geometry of the continuum medium will be achieved by the collection of the finite set of portions forming a non-overlapping domain. A FEM mesh is shown in figure (7).

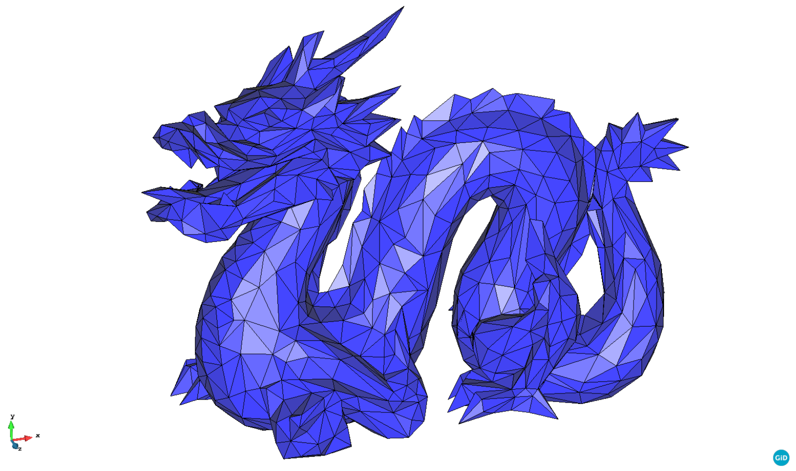


Figure 7: Stanford Dragon mesh (WRL file retrieved from Stanford University)

The solution variables of the finite element problem are expressed in terms of a polynomial expansion. The need of FEM comes from the fact that solving the analytical variation parameters is very complex in many cases, providing an approximation of the exact solution.

This section is based on the Chapter 1 and Chapter 2 of "*Structural Analysis with the Finite Element Method*" found in the references [8], with a slight change of notation and preserving the equations format.

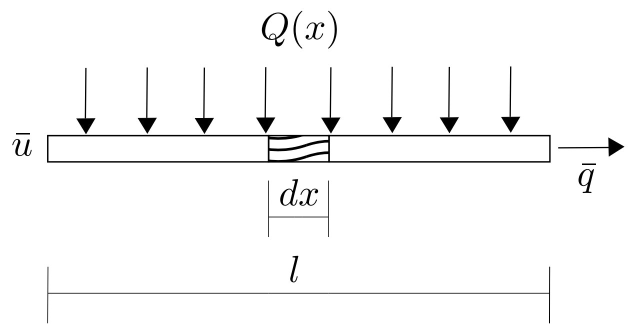
3.1 Analysis

To give a basic introduction of the methodology and concept of FEM, a simple FEM of a 1D Poisson equation is studied.

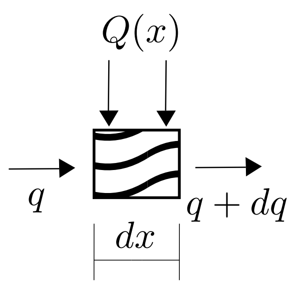
The simplest 1D governing equation for the Poisson problem is:

$$k\frac{d^2u}{dx^2} + Q(x) = 0 \tag{28}$$

Where u is the unknown variable (e.g. displacement), k represents a physical parameter (e.g.. stiffness), $Q(x)$ is the so-called source term (e.g. forces). In figure 8a, it is shown a graphic representation of the 1D problem, and in figure 8b, the infinitesimal representation.



(a) Representation of a Poisson bar problem



(b) Infinitesimal section of bar

Figure 8: Poisson problem for a bar (Redrawn from Oñate)

3.2 Weighted Residual Method

The weighted residual method (WRM) is a transformation of the governing equations to an equivalent integral expression.

Let $A(u)$ be the governing equation in 28:

$$A(u) = \frac{d}{dx}\left(k\frac{du}{dx}\right) + Q = 0 \quad \text{in} \quad \Omega \tag{29}$$

where Ω is the domain in analysis.
And, let $B(u)$ be the equation expressing the boundary conditions of the differential equation $A(u)$:

$$B(u) : \begin{cases} u - \bar{u} = 0 & \text{in } \Gamma_u \\ k\dfrac{du}{dx} + \bar{q} = 0 & \text{in } \Gamma_q \end{cases} \tag{30}$$

where Γ_u is the Dirichlet boundary where the unknown function is prescribed (e.g. prescribed displacement) and Γ_q is the Neumann boundary (e.g. prescribed forces). A diagram of a sample domain with boundary conditions is shown in figure 9. The unknown u and the parameters k, Q, and q have different meanings depending on the physical problem (e.g. structural or heat problem). The equivalent integral expression of the differential equation is obtained by

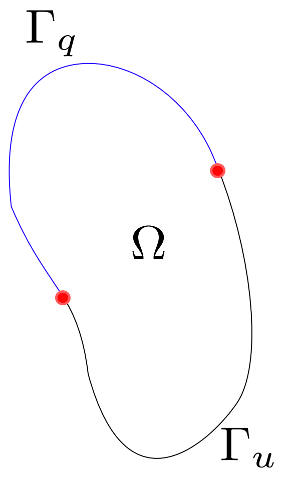


Figure 9: Domain (Ω), Dirichlet boundary (Γ_u), and Neumann boundary (Γ_q) representation.

multiplying $A(u)$ and $B(u)$ by arbitrary weighting functions $W(x)$ and $\bar{W}(x)$ respectively:

$$W(x)A(u) + \bar{W}(x)B(u) = 0 \tag{31}$$

And then integrating over the analysis domain (including the boundary):

$$\int_{\Omega} W(x)A(u)d\Omega + \oint_{\Gamma} \bar{W}(x)B(u)d\Gamma = 0 \tag{32}$$

The interesting feature of equation 32, is that it can be expressed in terms of a sum of integrals of the finite elements because of the additive property of the integral:

$$\sum_{e} \int_{\Omega^e} W(x)A(u)d\Omega + \sum_{e} \oint_{\Gamma^e} \bar{W}(x)B(u)d\Gamma = 0 \tag{33}$$

This equation 33 is the basis of the FEM assembly analysis.

3.3 Unknown variable

As previously mentioned, FEM is going to give an approximation of the analytical unknowns. The congruent approximated value is:

$$u(x,t) \cong \hat{u}(x,t) \tag{34}$$

The approximation of the unknowns ($\hat{u}(x,t)$) is usually expressed as a linear combination of the so-called shape functions $N_i(x)$ and the unknown parameters $a_i(t)$ as follows:

$$\hat{u}(x,t) = \sum_{i=0}^{n} N_i(x)a_i(t) \tag{35}$$

Substituting 34 into 32:

$$\int_{\Omega} W(x)A(\hat{u})d\Omega + \oint_{\Gamma} \bar{W}(x)B(\hat{u})d\Gamma = 0 \tag{36}$$

This expression is called the *weighted residual expression* because $A(\hat{u})$ and $B(\hat{u})$ represent the residuals of the approximation of the solution in the domain and its boundary. In such a way that:

$$\begin{aligned} A(\hat{u}) &= r_{\Omega} \quad \text{in } \Omega \\ B(\hat{u}) &= r_{\Gamma} \quad \text{in } \Gamma \end{aligned} \tag{37}$$

Where r_{Ω} and r_{Γ} are the residuals on the domain and boundary respectively.

Substituting 37 into 36:

$$\int_{\Omega} W r_{\Omega} d\Omega + \oint_{\Gamma} \bar{W} r_{\Gamma} d\Gamma = 0 \tag{38}$$

Such that if $u = \hat{u}$, the residuals $r_{\Omega} = r_{\Gamma} = 0$.

Finally, substituting 35 into 36, the following system is obtained:

$$\int_{\Omega} W_i A\left(\sum_{j} N_j a_j\right) d\Omega + \oint_{\Gamma} \bar{W}_i B\left(\sum_{j} N_j a_j\right) d\Gamma = 0; \quad i = 1, n \tag{39}$$

The equation 39 can simply be expressed in a matrix form, resulting into a linear system of equations:

$$\mathbf{Ku} = \mathbf{f} \tag{40}$$

Where $\mathbf{K} \in \mathbb{R}^{n \times n}$ contains geometrical and physical properties of the problem, $\mathbf{u} \in \mathbb{R}^n$ contains the unknown solution, and $\mathbf{f} \in \mathbb{R}^n$ contains the source term function values in the boundary.

4 Reduced Order Models (ROM)

The concept of model order reduction is old in structural mechanics, covering a wide range of purposes (e.g. designing and testing), in a way that leads to the creation of a basis that allows a cheap and fast comparison to experimental and simulation results. It is often associated to very high-dimensional finite element models. They capture the behavior of the complex models to later quickly study a system's dominant effects with high-fidelity results using low computational resources (e.g. ROM gives the possibility to predict the behaviour of a multibody system within a micro time-step [9]).

Throughout this paper, the high-fidelity model will be a Finite Element model to which will be referred to as the "*Full Order Model*" (FOM), meanwhile the standard Galerkin projection-based reduced model will be referred as the "*Reduced Order Model"* (ROM). To obtain the dimensionality reduction of the model, it is needed to find a basis inside the FOM space, therefore, given its orthonormal nature, it is proposed to use the Proper Orthogonal Decomposition for the computation of such basis [10].
In figure 10 it is shown the equivalence of the spaces for the different models. Being τ the space of functions where the problem is defined and, τ_h and τ_r the subs-spaces of functions FOM and ROM respectively.

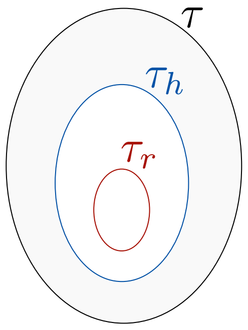


Figure 10: Illustration for τ and its sub-spaces in FOM and ROM.

4.1 Proper Orthogonal Decomposition

This section is an adaption from the Chapter 11 of [1], where some functions were adapted to this paper's purposes.

The application of the SVD algorithm to a partial differential equation is a dimensionality reduction technique which receives the name of "*Proper Orthogonal Decomposition*". The success of this implementation is that usually complex systems are governed by low-dimensional patterns of dynamic activity.

To simplify the idea, the 1D Poisson equation is considered to give the main idea of how the POD is implemented and applied to the FOM approximation.

One common technique to solve differential equations is the use of separation of variables:

$$\mathbf{u}(x,t) = \boldsymbol{\phi}(x)\mathbf{a}(t) \tag{41}$$

Where $\boldsymbol{\phi}$ and $\mathbf{a}$ characterize the spatial and time dependence respectively. The fact that the solution is not known *a priori*, leads to a typical assumption that the solution can be described as a sum of a set of basis modes which are constant coefficients and are used to construct $\boldsymbol{\phi}(x)$:

$$\mathbf{u}(x,t) = \sum_{k=1}^{r} \boldsymbol{\phi}_k(x) a_k(t) \tag{42}$$

where r stands for the number of modes. To proceed with the construction of an optimal POD modes, a *training stage* of the FOM has to be done, where the snapshot $\mathbf{u}_k$ consists of sample solutions of the FOM and the subscript k indicates the sample time-step:

$$\mathbf{u}_k := [\mathbf{u}(x_1,t_k), \mathbf{u}(x_2,t_k), \ldots, \mathbf{u}(x^n,t_k)]^{\mathbf{T}} \tag{43}$$

Building a time-steps series of data $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} | & | & & | \\ \mathbf{u}_1 & \mathbf{u}_2 & \ldots & \mathbf{u}_m \\ | & | & & | \end{bmatrix} \tag{44}$$

with m being the number of time-steps. The matrix $\mathbf{X}$ will be called from now on the Snapshot matrix.

As previously discussed in the "Singular Value Decomposition" section, the SVD provides a unique matrix decomposition:

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^{\mathbf{T}} \tag{45}$$

And applying the rSVD algorithm previously explained in "Randomized Singular Value Decomposition" subsection, it can be obtained a truncated low-rank decomposition of the SVD:

$$\tilde{\mathbf{X}} = \tilde{\mathbf{U}}\tilde{\boldsymbol{\Sigma}}\tilde{\mathbf{V}}^{\mathbf{T}} \tag{46}$$

where $||\mathbf{X} - \tilde{\mathbf{X}}||_F^2 < \epsilon ||\mathbf{X}||_F^2$. This decomposition results into the construction of the modes $\boldsymbol{\phi}_k$ coming from the left singular vectors $\tilde{\mathbf{U}}$:

$$\tilde{\mathbf{U}} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\phi}_1 & \boldsymbol{\phi}_2 & \ldots & \boldsymbol{\phi}_r \\ | & | & & | \end{bmatrix} \tag{47}$$

where r stands for the truncated modes. It is important to observe that this modes represent a data-driven model that provides an equation free analysis (this is crucial for the reconstruction of the stresses and reactions that will be presented in the following chapters).

It is now possible to obtain an approximation of the FOM solution 35:

$$\underbrace{\mathbf{u}(x,t)}_{\textbf{FOM}} \cong \underbrace{\tilde{\mathbf{u}}(x,t)}_{\textbf{ROM}} \tag{48}$$

Where the Galerkin projection onto the POD modes yields:

$$\tilde{\mathbf{u}}(x,t) = \sum_{i=1}^{r} \boldsymbol{\phi}_i(x) q_i(t) \tag{49}$$

In matrix form:

$$\tilde{\mathbf{u}} = \tilde{\mathbf{U}}\mathbf{q} \tag{50}$$

where $\tilde{\mathbf{U}} \in \mathbb{R}^{n\times r}$ and $\mathbf{q} \in \mathbb{R}^r$. Where n is the state-dimension, and r the truncated modes.

Substituting into the matrix form 40 of the governing equation and multiplying by $\tilde{\mathbf{U}}^{\mathbf{T}}$ gives the reduced order matrix form:

$$\tilde{\mathbf{U}}^{\mathbf{T}}\mathbf{K}\tilde{\mathbf{U}}\mathbf{q} = \tilde{\mathbf{U}}^{\mathbf{T}}\mathbf{f} \tag{51}$$

4.2 Hyper Reduced Order Models (HROM)

Let us focus now on the problem of how to efficiently compute the reduced vectors of the elemental residuals:

$$\tilde{\mathbf{U}}^{\mathbf{T}}\mathbf{f} = \sum_{e=1}^{n_e} \tilde{\mathbf{U}}_e^{\mathbf{T}}\mathbf{f}_e(\tilde{\mathbf{U}}_e\mathbf{q}) \tag{52}$$

This implies that the cost of building the ROM could be as high as for building the FOM, arising the need to implement one of the so-called "*Hyper-reduction techniques*". This techniques rely on a careful selection of a subset of elements (e_s) with a respective weight such that:

$$\tilde{\mathbf{U}}^{\mathbf{T}}\mathbf{f} = \sum_{e=1}^{n_e} \tilde{\mathbf{U}}_e^{\mathbf{T}}\mathbf{f}_e(\tilde{\mathbf{U}}_e\mathbf{q}) \cong \sum_{e \in e_s} \tilde{\mathbf{U}}_e^{\mathbf{T}}\mathbf{f}_e(\tilde{\mathbf{U}}_e\mathbf{q}) \underbrace{\omega_e}_{weight} \tag{53}$$

4.2.1 Empirical Cubature Method

To find the subset of elements e_s it can be used the method proposed in [11], and further refined in [12], called the *Empirical Cubature Method.*

In a nutshell, the Empirical Cubature Method implies saving the training's projected elemental residuals $\mathbf{R}_k^{\phi}$ into a snapshot matrix:

$$\mathbf{S}_r = [\mathbf{R}_1^{\phi}, \mathbf{R}_2^{\phi}, \cdots, \mathbf{R}_m^{\phi}] \tag{54}$$

Find a modal basis by applying the SVD:

$$\mathbf{S}_r = \underbrace{\mathbf{U}\boldsymbol{\Sigma}}_{\mathbf{G}^{\mathbf{T}}} \mathbf{V}^{\mathbf{T}} \tag{55}$$

Note that $\mathbf{G}$ is associated to the elements and modes.
Minimize an objective function of the form:

$$\mathbf{J}(\mathbf{G}, e_s, \boldsymbol{w}) = ||\mathbf{G1} - \mathbf{G}_{e_s}\boldsymbol{w}||_2^2 \tag{56}$$

Where s_e and $\boldsymbol{w}$ are the selected elements and non-negative elemental weights such that:

$$||\mathbf{G1} - \mathbf{G}_{e_s}\boldsymbol{w}||_2^2 < \epsilon \tag{57}$$

4.3 Gappy Data

This section arise from the fact that many times, it will be necessary to restore incomplete or degraded information coming from an image, data set, and/or the solution of a variable in a computational mechanics simulation. In previous sections, it has been shown how to recover a large data set or matrix with the most significant information coming from the SVD factorization. The question is, is it possible to recover a *snapshot* with incomplete or degraded information with the help of the *training stage (Snapshot Matrix)* ? How much information is required to recover the *snapshot* or how well the *space of training* has to be defined?

To illustrate and explain a solution to this problem, the same notation as from the previous chapters is going to be considered, where the *snapshot* is $\mathbf{u}$ and the *snapshot matrix* is $\mathbf{X}$.

When the *snapshot* is missing information or is only measured in certain points, a concept called a *masked snapshot* $\check{\mathbf{u}}$ is introduced, and it is defined as:

$$\check{\mathbf{u}} = \mathbf{M}\mathbf{u} \tag{58}$$

where $\mathbf{M} \in \mathbb{R}^{g\times n}$, $\mathbf{u} \in \mathbb{R}^{n}$, and $\check{\mathbf{u}} \in \mathbb{R}^{g}$. n and g stand for the state dimension and the *gappy* data measurements respectively.

The matrix $\mathbf{M}$ is a Boolean matrix that stores the information of where the gappy data is. Where $M_{ij} = 0$ on the mask and $M_{ij} = 1$ elsewhere. An example of $\mathbf{M}$ is shown below:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{59}$$

Applying the standard POD to equation 58:

$$\check{\mathbf{u}} \approx \mathbf{M}\sum_{i=1}^{r} \boldsymbol{\phi}_i q_i \tag{60}$$

Or in matrix form:

$$\check{\mathbf{u}} \approx \mathbf{M}\tilde{\mathbf{U}}\mathbf{q} \tag{61}$$

where $\check{\mathbf{u}} \in \mathbb{R}^{g}$, $\tilde{\mathbf{U}} \in \mathbb{R}^{n\times r}$, and $\mathbf{q} \in \mathbb{R}^{r}$. r stands for the number of truncated SVD modes. Everson and Sirovich [13] showed that $\mathbf{q}$ minimizes the error in:

$$||\check{\mathbf{u}} - \mathbf{M}\tilde{\mathbf{u}}||_2^2 \quad \rightarrow \quad ||\check{\mathbf{u}} - \mathbf{M}\tilde{\mathbf{U}}\mathbf{q}||_2^2 \tag{62}$$

And therefore, recalling equation 50, considering a masked solution:

$$\underbrace{\mathbf{M}\tilde{\mathbf{u}}}_{\check{\mathbf{u}}} = \underbrace{\mathbf{M}\tilde{\mathbf{U}}}_{\check{\mathbf{U}}}\mathbf{q} \tag{63}$$

where $\check{\mathbf{u}} \in \mathbb{R}^{g}$ and $\check{\mathbf{U}} \in \mathbb{R}^{g\times r}$.

Solving 63 for $\mathbf{q}$ by pre-multiplying by the pseudo-inverse of $\check{\mathbf{U}}$:

$$\underbrace{\check{\mathbf{U}}^{\dagger}\check{\mathbf{U}}}_{\mathbf{I}}\mathbf{q} = \check{\mathbf{U}}^{\dagger}\check{\mathbf{u}} \tag{64}$$

yields:

$$\mathbf{q} = \check{\mathbf{U}}^{\dagger}\check{\mathbf{u}} \tag{65}$$

The approximation of the FOM solution with gappy data is achieved by substituting 65 into 50:

$$\tilde{\mathbf{u}} \approx \tilde{\mathbf{U}}\check{\mathbf{U}}^{\dagger}\check{\mathbf{u}} \tag{66}$$

where $\tilde{\mathbf{u}} \in \mathbb{R}^{n}$, $\tilde{\mathbf{U}} \in \mathbb{R}^{n\times r}$, $\check{\mathbf{U}}^{\dagger} \in \mathbb{R}^{r\times g}$, and $\check{\mathbf{u}} \in \mathbb{R}^{g}$.

4.4 Static condensation

The *static condensation* is a ROM technique that represents a FOM model only by means of the prescribed boundary condition (master) degrees of freedom, by expressing the non-prescribed boundary condition degrees (slave) of freedom in terms of the master.

Considering the static equation in 40:

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{67}$$

where for a structural mechanics problem, $\mathbf{K} \in \mathbb{R}^{n \times n}$ is the stiffness matrix, $\mathbf{u} \in \mathbb{R}^n$ is the displacement vector, and $\mathbf{f} \in \mathbb{R}^n$ is the force vector. n is the number of degrees of freedom of the system. Now, considering that the problem can be separated into master and slave degrees of freedom:

$$\begin{bmatrix} \mathbf{K_{mm}} & \mathbf{K_{ms}} \\ \mathbf{K_{sm}} & \mathbf{K_{ss}} \end{bmatrix} \begin{pmatrix} \mathbf{u_m} \\ \mathbf{u_s} \end{pmatrix} = \begin{pmatrix} \mathbf{f_m} \\ \mathbf{f_s} \end{pmatrix} \tag{68}$$

where $\mathbf{K_{mm}} \in \mathbb{R}^{n_m \times n_m}$, $\mathbf{K_{ms}} \in \mathbb{R}^{n_m \times n_s}$, $\mathbf{K_{sm}} \in \mathbb{R}^{n_s \times n_m}$, $\mathbf{K_{ss}} \in \mathbb{R}^{n_s \times n_s}$, $\mathbf{u_m} \in \mathbb{R}^{n_m}$, $\mathbf{u_s} \in \mathbb{R}^{n_s}$, $\mathbf{f_m} \in \mathbb{R}^{n_m}$, and $\mathbf{f_s} \in \mathbb{R}^{n_s}$. n_m and n_s are the master and slave degrees of freedom respectively, such that $n = n_m + n_s$.

Since for the slave degrees of freedom there is no restriction on it's displacement, the slave force vector $\mathbf{f_s} = \mathbf{0}$, yielding:

$$\begin{bmatrix} \mathbf{K_{mm}} & \mathbf{K_{ms}} \\ \mathbf{K_{sm}} & \mathbf{K_{ss}} \end{bmatrix} \begin{pmatrix} \mathbf{u_m} \\ \mathbf{u_s} \end{pmatrix} = \begin{pmatrix} \mathbf{f_m} \\ \mathbf{0} \end{pmatrix} \tag{69}$$

This system of equations can be partitioned into:

$$\mathbf{K_{mm}}\mathbf{u_m} + \mathbf{K_{ms}}\mathbf{u_s} = \mathbf{f_m} \tag{70}$$

$$\mathbf{K_{sm}}\mathbf{u_m} + \mathbf{K_{ss}}\mathbf{u_s} = \mathbf{0} \tag{71}$$

From equation 71, solving for the slave displacements in terms of the master displacements:

$$\mathbf{u_s} = -\mathbf{K_{ss}}^{-1}\mathbf{K_{sm}}\mathbf{u_m} \tag{72}$$

Substituting equation 72 into 70:

$$\mathbf{K_{mm}}\mathbf{u_m} + \mathbf{K_{ms}}(-\mathbf{K_{ss}}^{-1}\mathbf{K_{sm}}\mathbf{u_m}) = \mathbf{f_m} \tag{73}$$

Simplifying:

$$\underbrace{(\mathbf{K_{mm}} - \mathbf{K_{ms}}\mathbf{K_{ss}}^{-1}\mathbf{K_{sm}})}_{\mathbf{K_G}}\mathbf{u_m} = \mathbf{f_m} \tag{74}$$

The transformation matrix to map from the ROM into the FOM is then defined as:

$$\begin{bmatrix} \mathbf{u_m} \\ \mathbf{u_s} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} \\ -\mathbf{K_{ss}}^{-1}\mathbf{K_{sm}} \end{bmatrix}}_{\mathbf{T_G}} \begin{bmatrix} \mathbf{u_m} \end{bmatrix} \tag{75}$$

Yielding a ROM system of equations of the form:

$$\mathbf{K_G}\mathbf{u_m} = \mathbf{f_m} \tag{76}$$

where $\mathbf{K_G} \in \mathbb{R}^{n_m \times n_m}$.
Hence:

$$\mathbf{K_G} = \mathbf{T_G^T}\mathbf{K}\mathbf{T_G} \tag{77}$$

The resulting system of equations in equation 74 is equivalent to the system in equation 69, and was first proposed by Guyan in [14].

4.4.1 Craig-Bampton Method

The Guyan static condensation can be improved by considering a dynamic behaviour of the structure and accounting for the mass and stiffness matrix. Let us consider the equation of motion:

$$\mathbf{M\ddot{u}} + \mathbf{Ku} = \mathbf{f} \tag{78}$$

Recalling that $\mathbf{u}$ can be split into master and slave degrees of freedom such that:

$$\mathbf{u} = \begin{bmatrix} \mathbf{u_m} \\ \mathbf{u_s} \end{bmatrix} \tag{79}$$

The Craig-Bampton method [15] defines a transformation matrix such that:

$$\begin{bmatrix} \mathbf{u_m} \\ \mathbf{u_s} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{K_{ss}}^{-1}\mathbf{K_{sm}} & \boldsymbol{\phi_q} \end{bmatrix}}_{\mathbf{T_{CB}}} \begin{bmatrix} \mathbf{u_m} \\ \mathbf{q} \end{bmatrix} \tag{80}$$

where $\mathbf{T_{CB}} \in \mathbb{R}^{n \times n}$, $\boldsymbol{\phi_q} \in \mathbb{R}^{n_m \times n_s}$, and $\mathbf{q} \in \mathbb{R}^{n_s}$ are the Craig Bampton transformation matrix, the fixed base mode-shapes, and modal degrees of freedom respectively.

Substituting into equation 78 and pre-multiplying by $\mathbf{T_{CB}^T}$:

$$\underbrace{\mathbf{T_{CB}^T M T_{CB}}}_{\mathbf{M_{CB}}} \begin{bmatrix} \mathbf{\ddot{u}_m} \\ \mathbf{\ddot{q}} \end{bmatrix} + \underbrace{\mathbf{T_{CB}^T K T_{CB}}}_{\mathbf{K_{CB}}} \begin{bmatrix} \mathbf{u_m} \\ \mathbf{q} \end{bmatrix} = \mathbf{T_{CB}^T} \begin{bmatrix} \mathbf{f_m} \\ \mathbf{f_s} \end{bmatrix} \tag{81}$$

The Craig Bampton mass and stiffness matrices are then defined as:

$$\mathbf{M_{CB}} = \begin{bmatrix} \mathbf{M_{mm}} & \mathbf{M_{mq}} \\ \mathbf{M_{qm}} & \mathbf{M_{qq}} \end{bmatrix} \tag{82}$$

$$\mathbf{K_{CB}} = \begin{bmatrix} \mathbf{K_{mm}} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_{qq}} \end{bmatrix} \tag{83}$$

Recalling that the force vector $\mathbf{f_s} = 0$ and substituting 82 and 83 into 81:

$$\begin{bmatrix} \mathbf{M_{mm}} & \mathbf{M_{mq}} \\ \mathbf{M_{qm}} & \mathbf{M_{qq}} \end{bmatrix} \begin{bmatrix} \mathbf{\ddot{u}_m} \\ \mathbf{\ddot{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{K_{mm}} & \mathbf{0} \\ \mathbf{0} & \mathbf{K_{qq}} \end{bmatrix} \begin{bmatrix} \mathbf{u_m} \\ \mathbf{q} \end{bmatrix} = \begin{bmatrix} \mathbf{f_s} \\ 0 \end{bmatrix} \tag{84}$$

5 Multibody systems

In a simple manner, it can be said that a general multibody system (MBS) embraces two main characteristics, namely: (i) mechanical components that describe large translational and rotational displacements and (ii) kinematic joints that impose some constraints or restrictions on the relative motion of the bodies. In other words, a multibody system encompasses a collection of rigid and/or flexible bodies inter-connected by kinematic joints and possibly some force elements [2].
The bodies that are part of a multibody system can be considered rigid or flexible:

- Rigid: when its deformations are so small that they do not affect the overall behavior of the body.

- Flexible: its deformations affect the overall behavior of the body.

In the three-dimensional space, the motion of a free rigid body can be fully described by using six generalized coordinates associated with the six degrees of freedom. In turn, when a body includes some amount of flexibility, it has six rigid degrees of freedom plus the number of generalized coordinates necessary to describe the deformations [16]. The multibody system

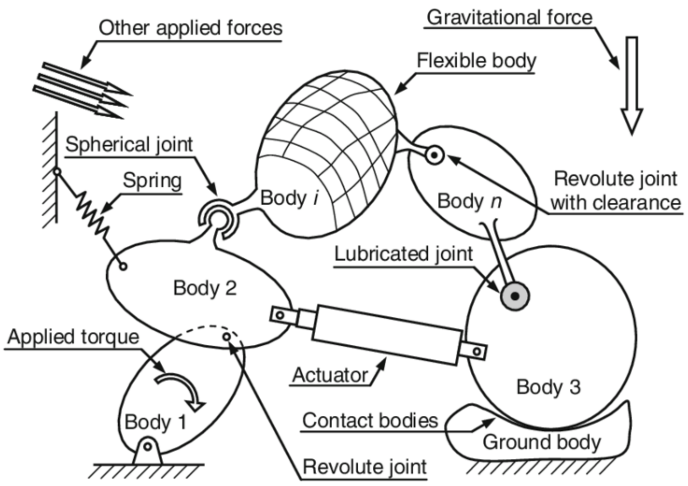


Figure 11: Abstract representation of a multibody system with its most significant components: bodies, joints, and forces elements (Image taken from [2])

solution can be very expensive, in such a way that it is necessary to make a reduction in the computational cost of the analysis, relying on a ROM technique.

5.1 Multi-freedom Constraints

The governing equations for constrained multibody systems are formulated in a manner suitable for their automation, a common way to enforce this, is via multifreedom constraints. Let us consider an example of an structural support, this conditions are imposed as constraints on individual degrees of freedom and are called “single-freedom” constraints (SFC). For example, let us consider a fixed node u_1:

$$u_{x1} = 0 \qquad u_{y1} = 0 \tag{85}$$

This condition involves two homogeneous SFCs, whereas if it was necessary to apply an offset whose value is different from “0” for the same node, it would involve two non-homogeneous SFCs.
In a more complex case, there are the so-called multifreedom constraints (MFC) which are equations that define and connect 2 or more constraints. In such a way that we can differentiate SFCs from MFCs in the following way:

$$\underbrace{N_d = P_v}_{\text{SFCs}} \qquad \underbrace{F(N_d) = P_v}_{\text{MFCs}} \tag{86}$$

Where:

- N_d = Nodal displacements
- P_v = Prescribed value

- $F(*)$ = functional equation

When nodal displacements involve more than one point, it is called multi-point [3]. For instance:

$$u_{x1} - 2u_{x2} = 1 \tag{87}$$

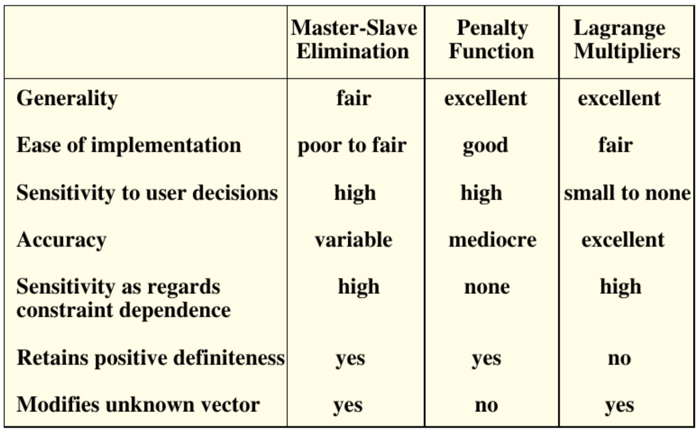
Both Mathworks and Kratos use this method to ensure that the interfaces to be connected in the multibody problem have 6 degrees of freedom represented in a single master node for each interface.

5.1.1 Imposing Methods

In order to take MFCs into account, the original system of equations must be modified by a change in the equations that define the master stiffness of the problem. This modification is called "*constraint application*" [3].
The three most used methods are:

- Master-Slave Elimination: The degrees of freedom involved in the MFC are separated into master and slave degrees of freedom, in such a way that the slave degrees of freedom are eliminated from the global system and only the master degrees of freedom appear.
- Penalty method: Each MFC is treated as a fictitious structural element that enforces the condition, being this parameterized with a numerical weight that will return to the original form when it tends to infinity.
- Lagrange Multiplier: For each MFC an unknown will be added to the initial system of equations in such a way that they will represent constraint forces that force the system to comply with the restrictions exactly as if they were applied to the unrestricted system.

In figure 12, a summary of the characteristics of these methods is shown, where it can be concluded that the Penalty method has a serious accuracy problem, Lagrange Multipliers despite its great characteristics such as generality, non-sensitivity to user decisions, among others, do not retain positive definiteness of the system, therefore, *Kratos Multihpyisics applies the method of Master-Slave elimination for MFCs.*



	Master-Slave Elimination	Penalty Function	Lagrange Multipliers
Generality	fair	excellent	excellent
Ease of implementation	poor to fair	good	fair
Sensitivity to user decisions	high	high	small to none
Accuracy	variable	mediocre	excellent
Sensitivity as regards constraint dependence	high	none	high
Retains positive definiteness	yes	yes	no
Modifies unknown vector	yes	no	yes

Figure 12: Assessment summary of three MFC application methods [3]

5.1.2 Master-Slave

The system of equations will be modified in such a way that when taking into account the different MFCs, a solution vector $\bar{\mathbf{u}}$ will be obtained from the elimination of the slave degrees of freedom of the original solution $\mathbf{u}$. This new vector $\bar{\mathbf{u}}$ will then have the master degrees of freedom and those that are not related to the MFCs [3].
This should be done with the help of a transformation matrix in such a way that:

$$\mathbf{u} = \mathbf{T}\bar{\mathbf{u}} \tag{88}$$

Recalling the original system of equations in 40:

$$\mathbf{K}\mathbf{u} = \mathbf{f} \tag{89}$$

Applying the MFCs of the problem and pre-multiplying by $\mathbf{T}^{\mathbf{T}}$ the new global system will be of the form:

$$\bar{\mathbf{K}}\bar{\mathbf{u}} = \bar{\mathbf{f}} \tag{90}$$

Where:

$$\mathbf{T}^{\mathbf{T}}\mathbf{K}\mathbf{T} = \bar{\mathbf{K}} \tag{91}$$

$$\mathbf{T}^{\mathbf{T}}\mathbf{f} = \bar{\mathbf{f}} \tag{92}$$

PART III

METHODOLOGY

6 GiD

GiD is a user friendly pre and post-processor for numerical simulation of engineering and scientific phenomena. GiD offers a wide variety of possibilities within its framework:

- Geometrical modeling (CAD)
- Mesh generation
- Definition of analysis data
- Data transfer to analysis software
- Post-processing operations
- Visualization of results

The most used characteristics for this paper were the geometrical modeling, mesh generation, definition of analysis data, and data transfer to analysis software.

Figure 13: GiD logo

GiD creates the necessary input files for Kratos Multiphysics 7 to solve the mechanical simulation via a *problem type* (visit Kratos Multiphysics - GiD for more details).

7 Kratos Multiphysics

Kratos is a powerful finite element method (FEM) based framework for building multidisciplinary programs, providing multiple tools for applications of multiphyiscs problems allowing interaction between them. Kratos has implemented an interface in Python to define the main procedure which significantly improves the flexibility of the framework in time of use.
Its core is mainly developed in C++, which provides a faster implementation by introducing object oriented programming. Kratos is a FREE Open source (visit Kratos Multiphysics Repository) which means the main code and program structure is available and aimed to grow with the need of any user willing to expand it.

Figure 14: Kratos logo

In appendix F, an example of how to build a Kratos simulation is given with an application of a process. However, it is worth mentioning that to build and run a simulation within Kratos framework, is necessary to build 4 indispensable files:

- ***.mdpa**: File containing the mesh geometry.
- **ProjectParameters.json**: File containing the problem settings and BCs.
- **MainKratos.py**: Main script to run the simulation.
- ***Materials.json**: File containing the material properties.

7.1 ROM Application

Kratos Multiphysics provides an application to develop ROMs. On this paper, it is provided a short overview of the capabilities that Kratos provides for an Structural Mechanics problem from the ROM point of view.
The first step to build a ROM is the training stage, where the user has to define a space of training of which all the *snapshots* of the time-step solutions are saved into a *Snapshot Matrix*. As mentioned in the section 4.1, the construction of a basis of which the finite element equations are projected is achieved via the singular value decomposition of the *Snapshot Matrix*. This method is called the *Proper Orthogonal Decomposition*.

7.1.1 Creating the SVD basis

An example of how a displacement *Snapshot Matrix* is obtained in Kratos is shown below:

```python
""
# This method is executed in order to finalize the current step
def FinalizeSolutionStep(self):
    super().FinalizeSolutionStep()
    ArrayOfDisplacements = []
    for node in self._GetSolver().GetComputingModelPart().Nodes:
        ArrayOfDisplacements.append(node.GetSolutionStepValue(DISPLACEMENT_X,0))
        ArrayOfDisplacements.append(node.GetSolutionStepValue(DISPLACEMENT_Y,0))
        ArrayOfDisplacements.append(node.GetSolutionStepValue(DISPLACEMENT_Z,0))
    self.time_step_solution_container.append(ArrayOfDisplacements)

# This method is executed after the computations, at the end of the solution-loop
def Finalize(self):
    super.()Finalize()
    SnapshotMatrix=self.GetSnapshotsMatrix()

def GetSnapshotsMatrix(self):
    SnapshotMatrix = np.zeros((len(self.time_step_solution_container[0]),len(self.time_step_solution_container)))
    for i in range(len(self.time_step_solution_container)):
        Snapshot_i= np.array(self.time_step_solution_container[i])
        SnapshotMatrix[:,i] = Snapshot_i.transpose()
    return SnapshotMatrix
```

Once the *Snapshot Matrix* is saved, it is needed to apply the SVD to create a basis with a given tolerance:

```
""
u,s,_,_= RandomizedSingularValueDecomposition().Calculate(SnapshotMatrix,1e-6)
```

where u and s are the *left singular values* ($\mathbf{\Phi}$) and *singular values* respectively.
The *left singular values* have to be saved in a *json* format to read it when building the ROM.

7.1.2 Solving ROM

Kratos Multiphyisics' solution strategy is based on looping over the elements, assemble the global system, and solve it. The ROM application considers the same solution strategy by looping over the elements but reducing the left hand side (LHS) and right hand side (RHS) of the system with the SVD basis:

$$\underbrace{\mathbf{K} = \sum_e \mathbf{K}^e}_{\text{LHS FOM}} \qquad \underbrace{\mathbf{K}_{\text{ROM}} = \sum_e \mathbf{\Phi}^{e\text{T}}\mathbf{K}^e\mathbf{\Phi}^e}_{\text{LHS ROM}} \tag{93}$$

$$\underbrace{\mathbf{b} = \sum_e \mathbf{b}^e}_{\text{RHS FOM}} \qquad \underbrace{\mathbf{b}_{\text{ROM}} = \sum_e \mathbf{\Phi}^{e\text{T}}\mathbf{b}^e}_{\text{RHS ROM}} \tag{94}$$

This will result into a reduced system of equations of the form:

$$\mathbf{K}_{\text{ROM}}\mathbf{du}_{\text{ROM}} = \mathbf{b}_{\text{ROM}} \tag{95}$$

And solve for $\mathbf{du}_{\text{ROM}}$.
The next step is to project the ROM solution to the FOM fine basis, this is achieved by:

$$\mathbf{du} = \mathbf{\Phi}\mathbf{du}_{\text{ROM}} \tag{96}$$

In figure 15, a flowchart is presented.

7.1.3 HROM

Rom application also gives the possibility to build a Hyper Reduced Order Model by finding a set of elements and corresponding positive weights, such that when building the reduced system contributions the loop will only be held on the *selected elements* ($e_s << e$), obtaining an accurate approximation of the ROM model.

To find the set of elements, it is necessary to train the HROM (same training space used to train ROM) and build a *Snapshot Matrix* of the projected residuals (RHS). Then, the *Empirical Cubature Method* (ECM) [11] will select the elements and it's corresponding weights by minimizing a least-squares problem.

The solution strategy for the HROM models remains the same as for the ROM, but now

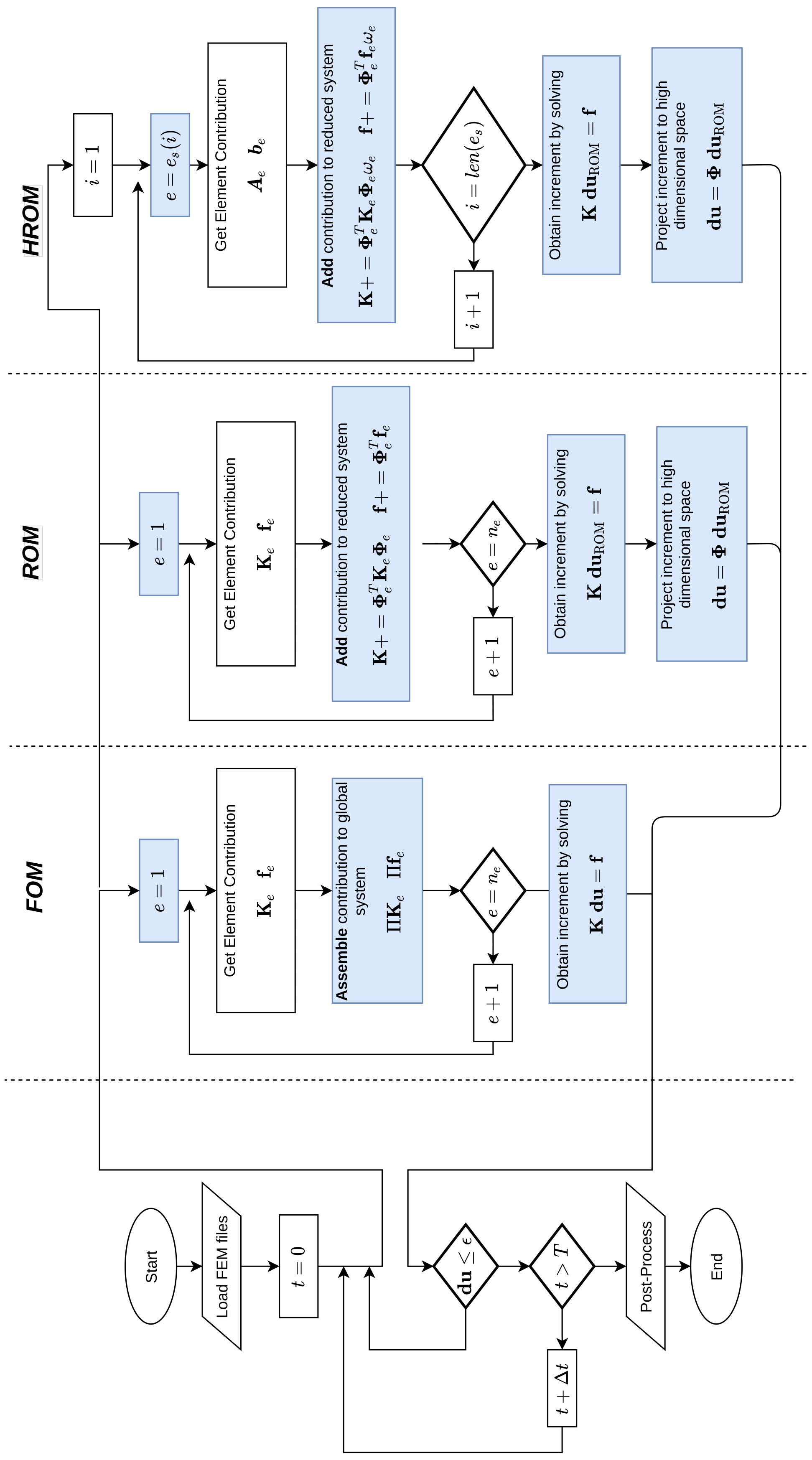

HROM
$i = 1$
$e = e_s(i)$
Get Element Contribution
$\boldsymbol{A}_e \;\; \boldsymbol{b}_e$
Add contribution to reduced system
$\mathbf{K}{+}= \boldsymbol{\Phi}_e^T \mathbf{K}_e \boldsymbol{\Phi}_e \omega_e \quad \mathbf{f}{+}= \boldsymbol{\Phi}_e^T \mathbf{f}_e \omega_e$
$i = len(e_s)$
$i + 1$
Obtain increment by solving
$\mathbf{K}\,\mathbf{du}_{\mathrm{ROM}} = \mathbf{f}$
Project increment to high dimensional space
$\mathbf{du} = \boldsymbol{\Phi}\,\mathbf{du}_{\mathrm{ROM}}$
ROM
$e = 1$
Get Element Contribution
$\mathbf{K}_e \;\; \mathbf{f}_e$
Add contribution to reduced system
$\mathbf{K}{+}= \boldsymbol{\Phi}_e^T \mathbf{K}_e \boldsymbol{\Phi}_e \quad \mathbf{f}{+}= \boldsymbol{\Phi}_e^T \mathbf{f}_e$
$e = n_e$
$e + 1$
Obtain increment by solving
$\mathbf{K}\,\mathbf{du}_{\mathrm{ROM}} = \mathbf{f}$
Project increment to high dimensional space
$\mathbf{du} = \boldsymbol{\Phi}\,\mathbf{du}_{\mathrm{ROM}}$
FOM
$e = 1$
Get Element Contribution
$\mathbf{K}_e \;\; \mathbf{f}_e$
Assemble contribution to global system
$\Pi\mathbf{K}_e \;\; \Pi\mathbf{f}_e$
$e = n_e$
$e + 1$
Obtain increment by solving
$\mathbf{K}\,\mathbf{du} = \mathbf{f}$
Start
Load FEM files
$t = 0$
$\mathbf{du} \leq \epsilon$
$t > T$
$t + \Delta t$
Post-Process
End

Figure 15: Kratos Implementation Flowchart

only running a loop over the *selected elements*, project the contributions to the reduced basis and multiply the contribution by the corresponding weight obtained on the ECM:

$$\underbrace{\mathbf{K}_{\text{HROM}} = \sum_{e \in e_s} \mathbf{\Phi}^{e\text{T}} \mathbf{K}^e \mathbf{\Phi}^e \omega_e}_{\text{LHS HROM}} \tag{97}$$

$$\underbrace{\mathbf{b}_{\text{HROM}} = \sum_{e \in e_s} \mathbf{\Phi}^{e\text{T}} \mathbf{b}^e \omega_e}_{\text{RHS HROM}} \tag{98}$$

In figure 15, a flowchart is presented.

7.2 Displacement Reconstruction

7.2.1 Stanford Bunny

To show the ROM application capabilities for a structural mechanics problem, the Stanford Bunny (SB) (WRL file retrieved from Stanford University) benchmark problem will be used, where a force will be applied to the face of the bunny and the base is fixed to undergo large displacements under a linear behaviour, the idea is best pictured in figure 16. The Stanford Bunny mesh has 49654 tetrahedral elements and 10915 nodes.

7.2.1.1 Results

The training of the ROM was done with a tolerance of 1e-8 for the SVD, leading to a truncation of 6 modes. In figure 17a it is shown the displacement field of the FOM model, in 17b it can be noticed that the solution looks exactly the same for the FOM model, and for the HROM, in figure 17c it is shown the projection of the HROM displacement field onto the skin of the FOM, which is taken only from 20 elements and 110 nodes shown in figure 17d (the opaque displacement field is shown only for topological reference). In addition, in figure 18a it can be observed a fast decay on the singular values and figure 18b shows that the first mode contains more than 99% of the information needed because it is a linear problem.

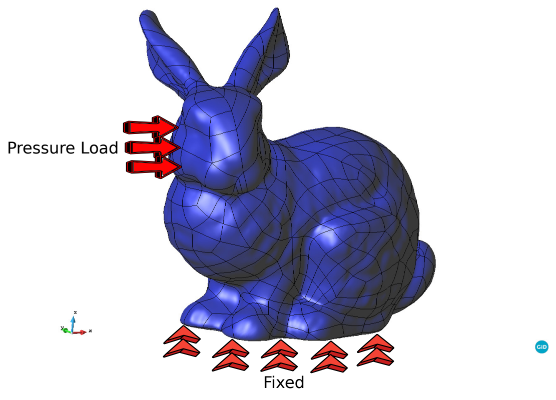
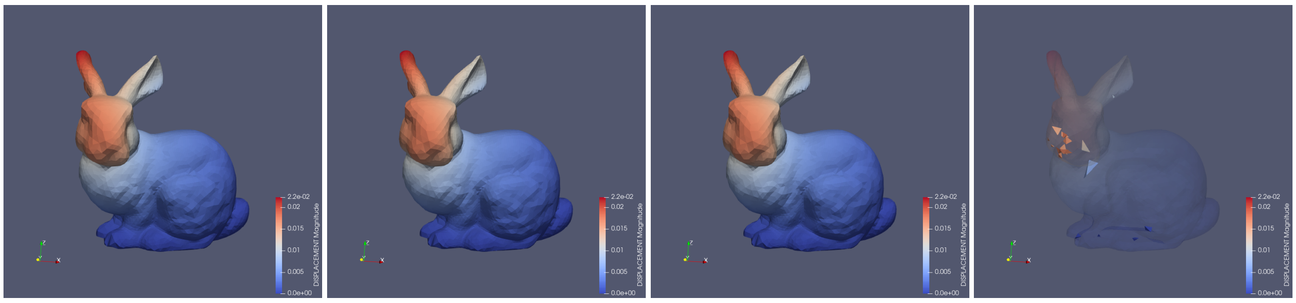


Figure 16: Stanford Bunny Diagram

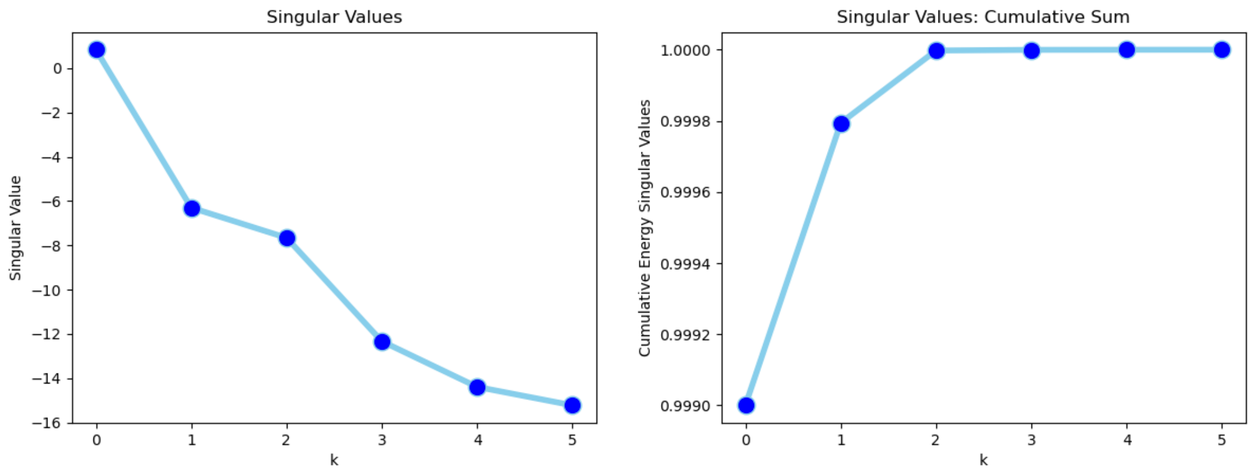
Tolerance	Modes	L_2 Error ROM	L_2 Error HROM
1.00E-02	1	1.80E-03	9.99E-02
1,00E-05	3	4.16E-06	3.17E-05
1,00E-06	4	9.68E-07	1.92E-05
1,00E-08	6	5.10E-08	1.80E-05
1.00E-09	8	5.17E-09	1.73E-05

Table 1: L_2 error for different truncated modes



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 17: FOM,ROM, and HROM SB displacements



(a) Singular Values (b) Cumulative Energy Singular Values

Figure 18: Displacement SB singular values

In table 1, it can be observed that the tolerance given for the SVD outputs an L_2 error of similar magnitude. If the tolerance leads to high truncated modes, the SVD will introduce noise to the approximation an the L_2 error will grow.

7.2.1.2 Kratos Methodology

To create the displacement basis to build the ROM, and run the HROM simulations it is needed to follow the extra steps:

- Add "*rom_basis_process*" to the project parameters with a given tolerance as shown in appendix A.

- Define a training space in the project parameters.
- Include the "*Empirical Cubature*" Rom Application strategy within the *Main_Kratos.py* file (refer to appendix B).
- Define an HROM project parameters and apply the conditions to the new model parts within the *Hyper Reduced Model Part.* Run HROM simulation as in appendix C.

7.2.2 Modal Basis

To have a physical point of view of the modes coming from the POD, an example of an structural frame being affected by a pressure load coming from the slab as shown in figure 19. In figure 21, it can clearly be observed how well the physics of the problem are projected into the modes. And in figure 20, it can clearly be observed that the first 7 modes cover more than 99% of the physical behaviour of the training.

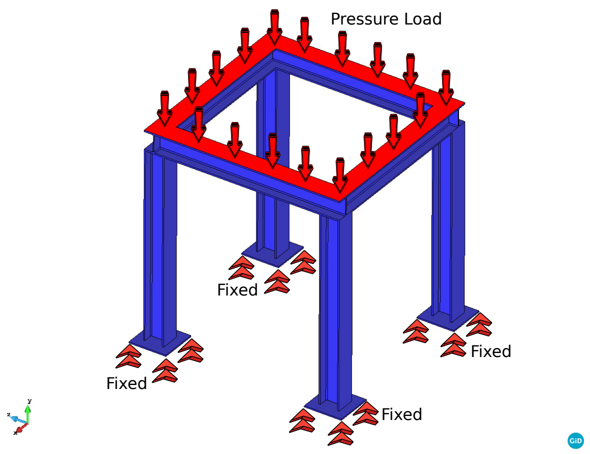
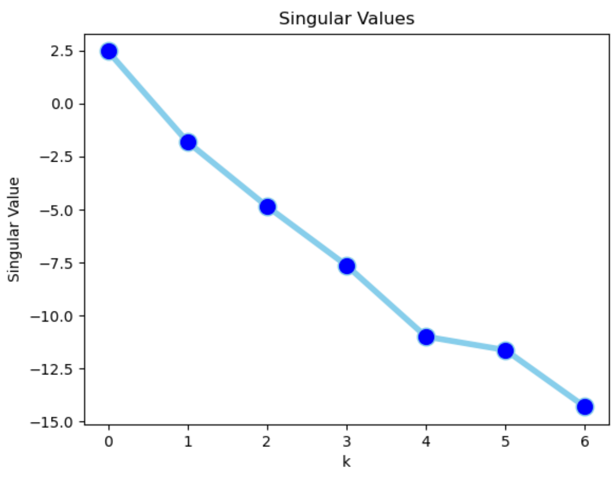
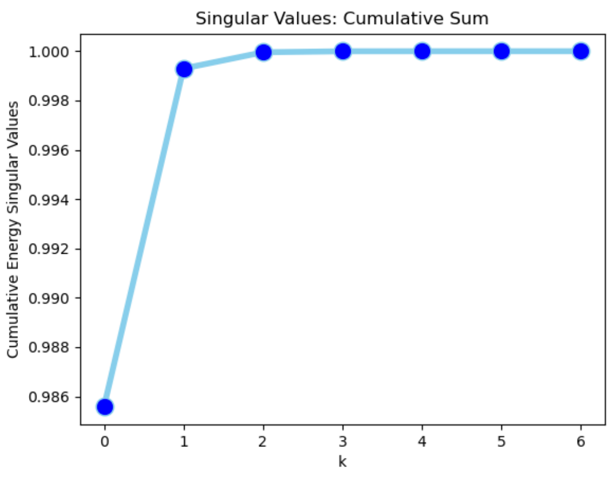


Figure 19: Structural Frame Diagram

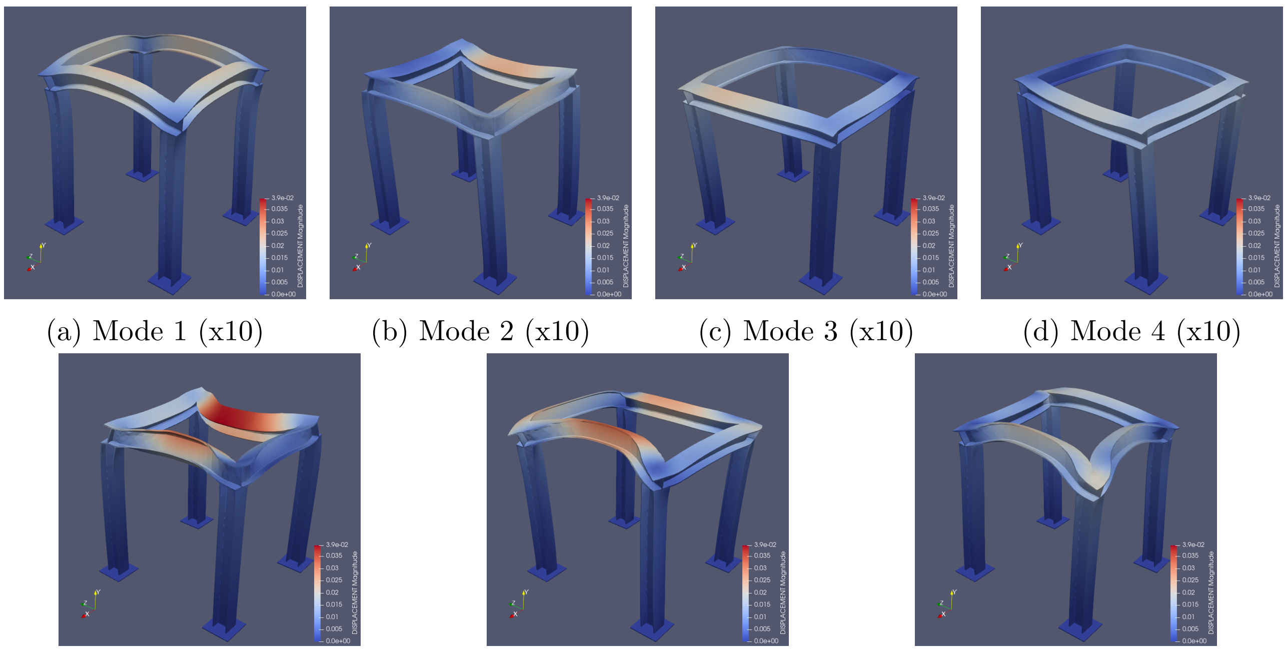


(a) Singular Values



(b) Cumulative Energy Singular Values

Figure 20: Structural Frame Singular Values



(a) Mode 1 (x10) (b) Mode 2 (x10) (c) Mode 3 (x10) (d) Mode 4 (x10)

(e) Mode 5 (x10) (f) Mode 6 (x10) (g) Mode 7 (x10)

Figure 21: Structural Frame POD modes

7.3 Stress Reconstruction

When building the HROM, the only variable that is reconstructed and projected to the fine basis is the displacement. Nevertheless, from the HROM, the information of the 2nd *Piola Kirchhoff* stress vector evaluated on the Gauss points can still be obtained on the selected elements.

$$\underset{e\in e_s}{\tilde{\boldsymbol{\sigma}}^{gp}} \in \mathbb{R}^{(g_e * n_{e_s} * s_d)} \tag{99}$$

where g_e is the number of Gauss points per element (for a domain with equal elemental characteristics), n_{e_s} is the number of selected elements, and s_d is the dimension of the stress field.

Recalling section 4.3, the selected elements coming from the HROM can be considered a masked subset approximation of elements of the FOM, such that:

$$\tilde{\boldsymbol{\sigma}}^{gp} \approx \mathbf{M}\boldsymbol{\sigma}^{gp} \tag{100}$$

where $\boldsymbol{\sigma}^{gp} \in \mathbb{R}^{(g_e * n_e * s_d)}$ is the snapshot of the 2nd *Piola Kirchhoff* stress vector evaluated on the Gauss points, n_e is the number of elements.

This means that the POD can be applied to the stress field similar to 61, such that there exists a linear operator $\boldsymbol{\Psi}$ and a coefficients vector $\mathbf{s}$ that minimizes:

$$||\boldsymbol{\sigma}^{gp} - \boldsymbol{\Psi}\mathbf{s}||_2^2 \tag{101}$$

where $\boldsymbol{\Psi}$ is the modal basis of left singular values coming from the SVD of the Snapshot Matrix of the 2nd *Piola Kirchhoff* stress vector evaluated on the Gauss points:

$$\mathbf{S}^{gp} = \underbrace{\mathbf{U}^{\boldsymbol{\sigma}}}_{\boldsymbol{\Psi}} \boldsymbol{\Sigma}^{\boldsymbol{\sigma}} \mathbf{V}^{\boldsymbol{\sigma}\mathbf{T}} \tag{102}$$

where $\mathbf{S}^{gp} \in \mathbb{R}^{(g_e * n_e * s_d) \times m}$, m is the number of time-steps of the training stage.

Setting 101 to $\mathbf{0}$:

$$\boldsymbol{\sigma}^{gp} = \boldsymbol{\Psi}\mathbf{s} \tag{103}$$

Recalling 100 by pre-multiplying 103 by a masking matrix $\mathbf{M}$:

$$\underbrace{\mathbf{M}\boldsymbol{\sigma}^{gp}}_{\tilde{\boldsymbol{\sigma}}^{gp}} = \underbrace{\mathbf{M}\boldsymbol{\Psi}}_{\substack{\tilde{\boldsymbol{\Psi}} \\ e \in e_s}}\mathbf{s} \tag{104}$$

Yielding:

$$\tilde{\boldsymbol{\sigma}}^{gp} = \tilde{\boldsymbol{\Psi}}\mathbf{s} \tag{105}$$

Pre-mulitplying by the pseudo-inverse of $\tilde{\boldsymbol{\Psi}}$ to solve for $\mathbf{s}$:

$$\mathbf{s} = \tilde{\boldsymbol{\Psi}}^{\dagger}\tilde{\boldsymbol{\sigma}}^{gp} \tag{106}$$

Substituting equation 106 into equation 103:

$$\boldsymbol{\sigma}^{gp} \approx \boldsymbol{\Psi}\tilde{\boldsymbol{\Psi}}^{\dagger}\tilde{\boldsymbol{\sigma}}^{gp} \tag{107}$$

yields the reconstruction of the stress field evaluated on the Gauss points.

7.3.1 Nodal Extrapolation (Smoothing)

The reconstructed stress field is:

$$\boldsymbol{\sigma}^{gp} \in \mathbb{R}^{(g_e * n_e * s_d)} \tag{108}$$

where $\boldsymbol{\sigma}^{gp}$ is a matrix containing the value at the Gauss points of the 2^{nd} *Piola Kirchhoff* stress vector of a rearranged sample time-step snapshot.
To visualize the stresses, it is needed to provide a continuous nodal field given by:

$$\boldsymbol{\sigma}^{n} = \bar{\mathbf{N}}\boldsymbol{\sigma}^{gp} \tag{109}$$

where $\boldsymbol{\sigma}^{n} \in \mathbb{R}^{(n_n * s_d)}$ and $\bar{\mathbf{N}}$, are the 2^{nd} *Piola Kirchhoff* stress vector evaluated on the nodes and the interpolation functions respectively (shape functions). n_n denotes the number of nodes.
The orthogonalization of the interpolation functions leads to:

$$\int_{\Omega_e} \bar{\mathbf{N}}^T\bar{\mathbf{N}}|\mathbf{J}|\mathbf{d\Omega_e}\boldsymbol{\sigma}^{n} = \int_{\Omega_e} \bar{\mathbf{N}}^T\boldsymbol{\sigma}^{gp}|\mathbf{J}|\mathbf{d\Omega_e} \tag{110}$$

where $|\mathbf{J}|$ is the determinant of the Jacobian [17].

However, since the ROM analysis is being performed in linear elements with only one Gauss point per element, the stress evaluated on the integration points will be constant through all the element's nodes, easing the extrapolation operation by only smoothing the stress field as a weighted average of the nodal areas defined as:

$$\boldsymbol{\sigma}^{n} = \mathbf{D}\mathbf{P}\boldsymbol{\sigma}^{gp} \tag{111}$$

where $\mathbf{D} \in \mathbb{R}^{n_n \times n_n}$, and $\mathbf{P} \in \mathbb{R}^{n_n \times n_e}$, are an inverted lumped nodal area matrix (diagonal) and the matrix of elemental area contribution to the nodes.
The inverse of the lumped nodal area matrix is defined as:

$$\mathbf{D}_{jj} = \frac{1}{\sum\limits_{k \in e_j} A_{jk}} \tag{112}$$

where A_{jk} is the k^{th} elemental area of the j^{th} neighbour node. e_j denotes the subset of neighbouring elements of the j^{th} node.
The matrix of elemental area contribution to the nodes is given by:

$$\mathbf{P}_{jk} = A_{jk} \tag{113}$$

where A_{jk} is the area of the neighboring k^{th} element of the j^{th} node.

7.3.2 Plasticity

When reconstructing a plasticity model, the modal basis can be better conditioned by performing an extra step. First, apply the SVD only to the Elastic training space (*Elastic Snapshot Matrix*):

$$\mathbf{S_e} = \underbrace{\mathbf{U_e}}_{\boldsymbol{\Psi_e}} \boldsymbol{\Sigma_e} \mathbf{V_e^T} \tag{114}$$

After building the elastic modal basis $\boldsymbol{\Psi_e}$, it is needed to build the *Elasto-plastic Snapshot Matrix* $\mathbf{S_{ep}}$ and perform an orthogonal projection onto the elastic basis:

$$\check{\mathbf{S}}_{\mathbf{ep}} = (\mathbf{I} - \boldsymbol{\Psi_e}\boldsymbol{\Psi_e}^{\mathbf{T}})\mathbf{S_{ep}} \tag{115}$$

looping on equation 115 may reduce numerical losses.

Applying the SVD to $\check{\mathbf{S}}_{\mathbf{ep}}$:

$$\check{\mathbf{S}}_{\mathbf{ep}} = \underbrace{\mathbf{U_{ep}}}_{\boldsymbol{\Psi_{ep}}} \boldsymbol{\Sigma}_{\mathbf{ep}} \mathbf{V_{ep}^T} \tag{116}$$

Once the plastic modal basis $\boldsymbol{\Psi_{ep}}$ is built, both modal basis can be appended column-wise:

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\Psi_e} & \boldsymbol{\Psi_{ep}} \end{bmatrix} \tag{117}$$

To ensure that the basis is orthogonal a *QR decomposition* can be performed to the elasto-plastic modal basis such that:

$$\boldsymbol{\Psi} = \underbrace{\mathbf{Q}}_{\check{\boldsymbol{\Psi}}} \mathbf{R} \tag{118}$$

This may cause extra modes and therefore, lead to more selected elements.

7.3.3 Kratos Methodology

To reconstruct the stress field for an HROM, it is needed to add some extra steps:

- Add the stress variable to reconstruct and the given tolerance to the "*rom_basis_process*" module as in appendix A.
- Include the "*stress_reconstruction_process*" module in the HROM project parameters as in appendix D.

7.4 Reaction Reconstruction

In structural mechanics, it is often needed to calculate the reactions of the structural elements on the restricted surfaces for design and analysis purposes. The Kratos' Structural Mechanics Application provides the residuals (RHS) of every element of the FOM, ROM, and HROM, and assembles the reactions as a sum of all the elemental contributions of the RHS to each of the nodes in the FOM. For the case of the ROM, it is only necessary to provide a new assembling of the reactions given the POD solution. In a nutshell, the assembling of the reactions was implemented by creating two lists of lists containing the neighboring elements for each node and the neighboring nodes for each element respectively and assigning a new reaction Id to map the corresponding contribution to the restricted surfaces. It is worth mentioning that the reactions will only be presented on the restricted surfaces, meaning that it is only needed to assemble the solution on the nodes that belong to these surfaces.

For the HROM, the provided information of the RHS yields only on the selected elements (e_s):

$$\underset{e\in e_s}{\tilde{\mathbf{r}}} \in \mathbb{R}^{(n_{e_s} * n_d)} \tag{119}$$

where $\tilde{\mathbf{r}}$ contains the RHS contribution of all the degrees of freedom of the selected elements subset. n_d is the number of degrees of freedom per element.

Similar to the stress field 100, $\tilde{\mathbf{r}}$, can be considered as masked approximated data from the FOM model:

$$\tilde{\mathbf{r}} \approx \mathbf{M}\mathbf{r} \tag{120}$$

where $\mathbf{r} \in \mathbb{R}^{(n_e * n_d)}$ is the FOM fine basis residuals vector.

Applying POD for the reconstruction of the residuals yields the minimization problem:

$$||\mathbf{r} - \mathbf{\Upsilon}\mathbf{z}||_2^2 \tag{121}$$

where $\mathbf{\Upsilon}$ is the modal basis of left singular values coming from the SVD of the Snapshot Matrix of the FOM residuals:

$$\mathbf{R} = \underbrace{\mathbf{U}^r}_{\mathbf{\Upsilon}} \mathbf{\Sigma}^r \mathbf{V}^{r\mathbf{T}} \tag{122}$$

where $\mathbf{R} \in \mathbb{R}^{(n_e * n_d)\times m}$.
Setting equation 121 to $\mathbf{0}$:

$$\mathbf{r} = \mathbf{\Upsilon}\mathbf{z} \tag{123}$$

Pre-multiplying by the masking matrix $\mathbf{M}$:

$$\underbrace{\mathbf{M}\mathbf{r}}_{\tilde{\mathbf{r}}} = \underbrace{\mathbf{M}\mathbf{\Upsilon}}_{\underset{e\in e_s}{\tilde{\mathbf{\Upsilon}}}}\mathbf{z} \tag{124}$$

Yielding:

$$\tilde{\mathbf{r}} = \tilde{\mathbf{\Upsilon}}\mathbf{z} \tag{125}$$

Pre-multiplying by the pseudo-inverse of $\tilde{\mathbf{\Upsilon}}$ to solve for $\mathbf{z}$:

$$\mathbf{z} = \tilde{\mathbf{\Upsilon}}^{\dagger}\tilde{\mathbf{r}} \tag{126}$$

To recover the FOM basis snapshot of residuals, substitute equation 126 into equation 123:

$$\mathbf{r} \approx \boldsymbol{\Upsilon}\tilde{\boldsymbol{\Upsilon}}^{\dagger}\tilde{\mathbf{r}} \tag{127}$$

Finally, it is needed to assemble the reaction on the restricted surfaces' nodes for the visualization of the post-process.

7.4.1 Kratos Methodology

To reconstruct the reaction field for a ROM or HROM, it is needed to add some extra steps:

- Add the stress variable to reconstruct and the given tolerance to the "*rom_basis_process*" module as in appendix A.
- Include the "*reaction_reconstruction_process*" module in the ROM/HROM project parameters as in appendix E.

8 Static condensation process

Recalling the Guyan static condensation in section 4.4, it seeks for a Stiffness Matrix that accounts only for the degrees of freedom involved in the boundary master nodes (interface nodes in a multibody problem):

$$\mathbf{K_G d_m} = \mathbf{f_m} \tag{128}$$

A ROM model can be obtained by training the FOM model, and through a surrogate model, obtain the Guyan's reduced stiffness matrix following these steps:

1. Identify the surfaces (interface frames) of the flexible body that will be the connections to the multibody or control model:

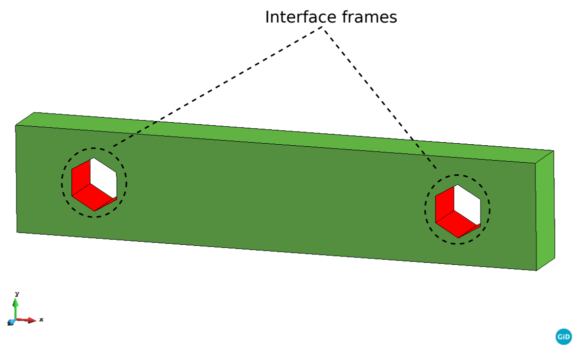


Figure 22: Static Condensation Interface Frames

2. Recalling that the static condensation is defined by master nodes, it is usual to propose a master node per interface frame at the center of the constrained surface:

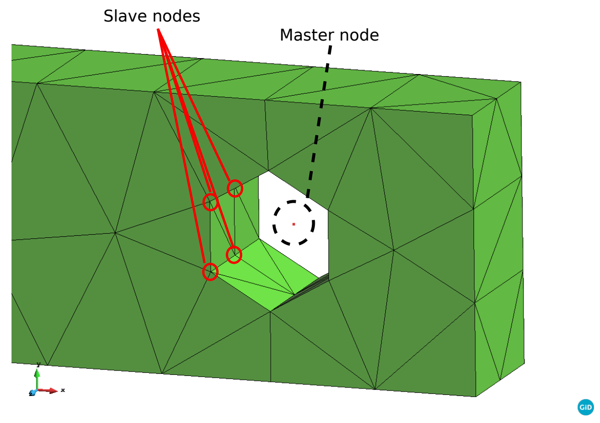


Figure 23: Static Condensation master and slave nodes

the slave nodes are the nodes that define the geometry of the interface frames.

3. To find the first column of the reduced stiffness matrix $\mathbf{K_G}$ of the reduced system:

$$\underbrace{\begin{bmatrix} K_{11} & K_{12} & \dots & K_{1m_d} \\ K_{21} & K_{22} & \dots & K_{2m_d} \\ \vdots & \vdots & \ddots & \vdots \\ K_{m_d1} & K_{m_d2} & \dots & K_{m_dm_d} \end{bmatrix}}_{\mathbf{K_G}} \underbrace{\begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_{m_d} \end{bmatrix}}_{\mathbf{d_m}} = \underbrace{\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{m_d} \end{bmatrix}}_{\mathbf{f_m}} \tag{129}$$

where m_d is the number of master degrees of freedom. It is needed to perform a FOM simulation imposing an infinitesimal strain $d_1 = \epsilon_1$ (small displacement) on the first master degree of freedom and fix all the other master degrees of freedom such that the reduced system is the following:

$$\begin{bmatrix} K_{11} & K_{12} & \dots & K_{1m_d} \\ K_{21} & K_{22} & \dots & K_{2m_d} \\ \vdots & \vdots & \ddots & \vdots \\ K_{m_d1} & K_{m_d2} & \dots & K_{dd} \end{bmatrix} \begin{bmatrix} \epsilon_1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{m_d} \end{bmatrix} \tag{130}$$

The equations will then yield:

$$\begin{pmatrix} K_{11}\epsilon_1 = f_1 \\ K_{21}\epsilon_1 = f_2 \\ \vdots \\ K_{m_d1}\epsilon_1 = f_{m_d} \end{pmatrix} \tag{131}$$

where ϵ_1, f_1,f_2,...,f_{m_d} are known. As mentioned in section 5.1, the infinitesimal displacements and rotations are imposed using *Multi-point Constraints*, via the *master-slave* imposing method 5.1.2. This can be repeated for all the degrees of freedom, $n \in m_d$:

$$\begin{pmatrix} K_{1n}\epsilon_n = f_1 \\ K_{2n}\epsilon_n = f_2 \\ \vdots \\ K_{m_dn}\epsilon_n = f_{m_d} \end{pmatrix} \tag{132}$$

where ϵ_n, f_1,f_2,...,f_{m_d} are known.

PART IV

RESULTS

9 Stress field

9.1 Wrench

To show the reconstruction of the stress field, a common benchmark test of a wrench is presented. Where torque is applied to turn an object such as fasteners, bolts, nuts, etc. In figure 24 a diagram of the proposed problem is shown. The material properties of the wrench are elastic, however, the training of the wrench will cover a geometrical non-linearity behaviour to test the behaviour under large displacements.

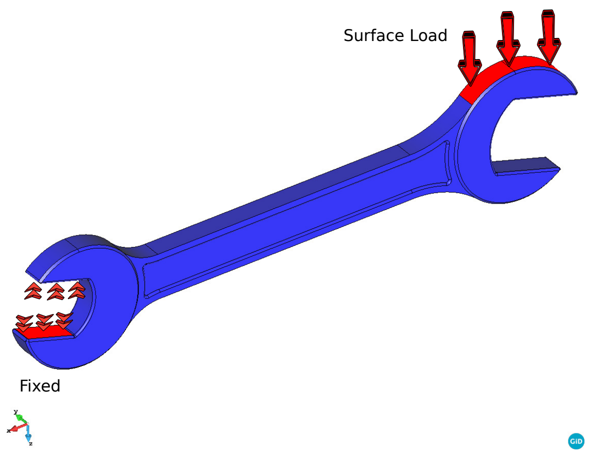


Figure 24: Wrench Diagram

9.1.1 Results

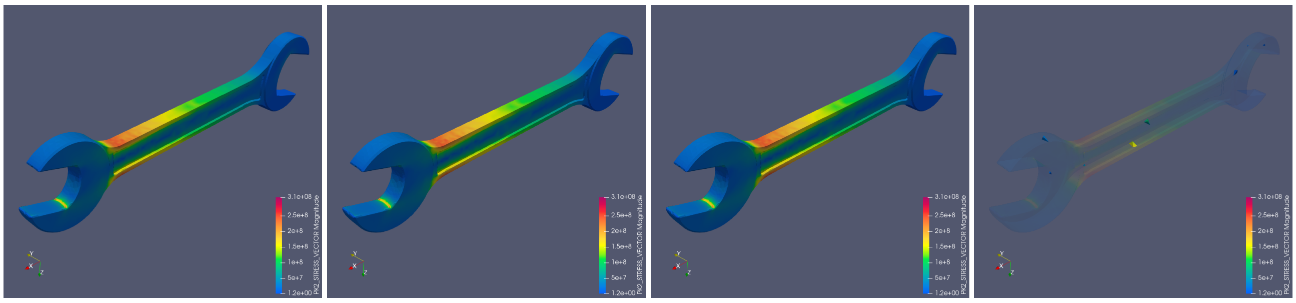
The model was successfully trained and built passing from 60425 elements to only 8 elements (4 modes). In figure 25a the stress field for the complete model is shown, in figure 25b the ROM model is shown with an obtained precision of 2.23E-07, and in figures 25c and 25d the stress field of the HROM are presented. The difference of the L_2 error for two different truncated modes can be found in tables 2 and 3.

Displacement			
Tolerance	Modes	L_2 Error ROM	L_2 Error HROM
1.00E-05	3	2.64E-06	3.36E-04
1.00E-06	4	1.49E-07	9.16E-05

Table 2: Wrench displacement field L_2 error for different truncated modes

Stress			
Tolerance	Modes	L_2 Error ROM	L_2 Error HROM
1.00E-04	3	4.56e-05	5.52E-04
1.00E-05	4	2.23E-07	2.97E-05

Table 3: Wrench stress field L_2 error for different truncated modes



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 25: FOM, ROM, and HROM Wrench Stresses

9.2 Uni-axial Tensile Test

The POD is to be tested through the uni-axial tensile test to check for the approximation of material non-linearities using the *Von Mises* plasticity model. This is a famous test that eliminates the dimensions of the geometry and capture the material properties by obtaining the stress-strain curve. The tensile test consists only in fixing the lower boundary of the geometry and apply a very slow prescribed displacement on the upper boundary as shown in figure 26.

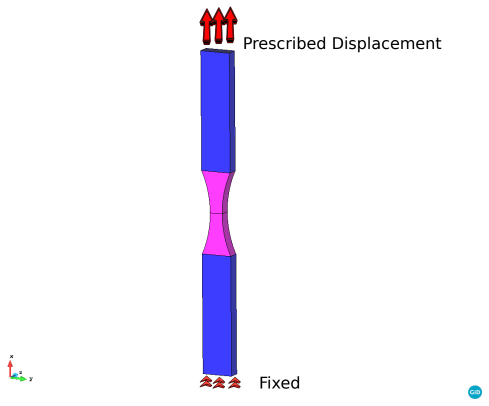
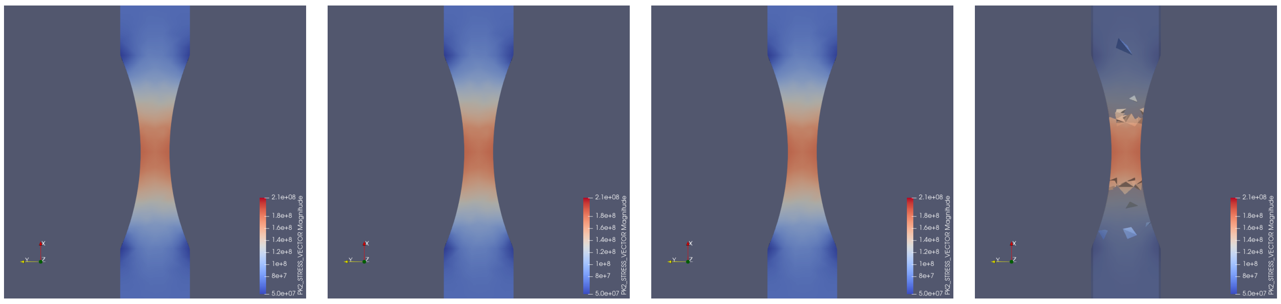


Figure 26: Tensile Test Diagram

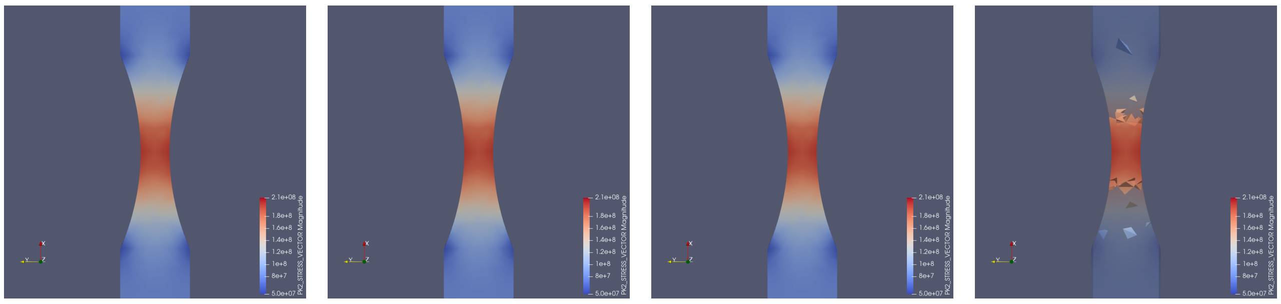
9.2.1 Results

The training space of the model was 210 simulations covering the elastic and plastic range. The testing space was the same as the training to check for consistency of the 210 simulations, covering the elastic yielding and hardening range. The resulting errors are presented in tables 4 and 5.



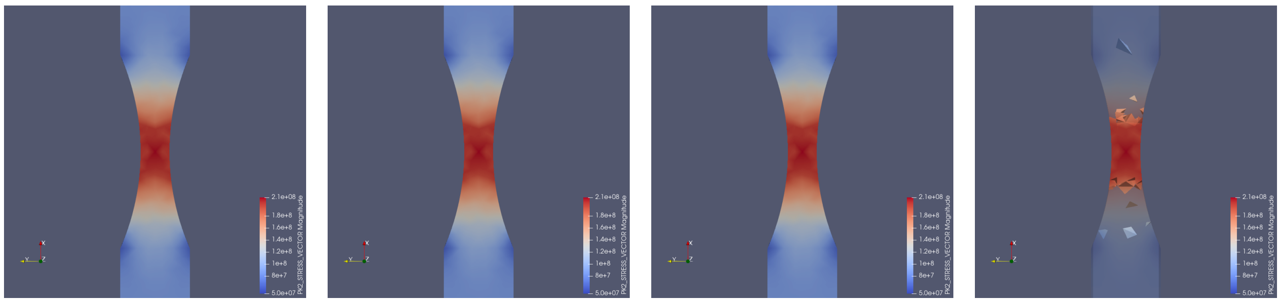
(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 27: FOM, ROM, and HROM Tensile Test Elasticity Range Stresses



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 28: FOM, ROM, and HROM Tensile Test Yielding Range Stresses



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 29: FOM, ROM, and HROM Tensile Test Hardening Range Stresses

Displacement		
Range	L_2 Error ROM	L_2 Error HROM
Elasticity	9.62E-08	3.04E-06
Yield	1.48E-07	3.07E-06
Hardening	4.39E-06	2.54E-05

Table 4: Tensile displacement field L_2 error for different ranges

Stress		
Range	L_2 Error ROM	L_2 Error HROM
Elasticity	1.30E-06	2.55E-05
Yield	2.83E-06	2.54E-05
Hardening	1.70E-05	2.84E-05

Table 5: Tensile displacement field L_2 error for different ranges

10 Reaction field

10.1 Rubber Fixed Ended Beam

In structural mechanics, it is often needed to know the reaction of the restricted surfaces or supports. A typical structural mechanics element is the beam, therefore, to test the reconstruction of the reaction field, a benchmark problem of a *fixed ended rubber beam* with two pressure loads is going to be reduced. In figure 30, it is provided a diagram of the problem definition, where the *pressure load 2* is bigger than the *pressure load 1*.

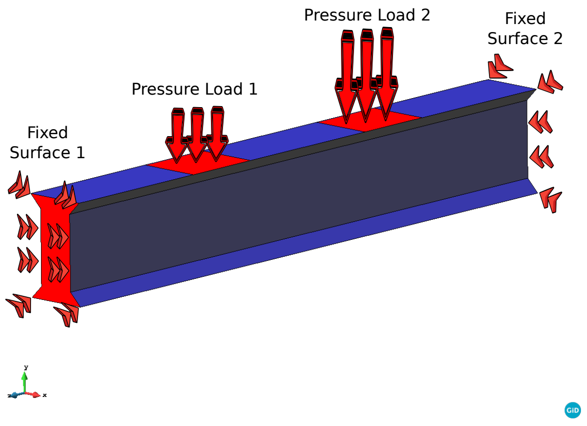


Figure 30: Fixed Ended Diagram

10.1.1 Results

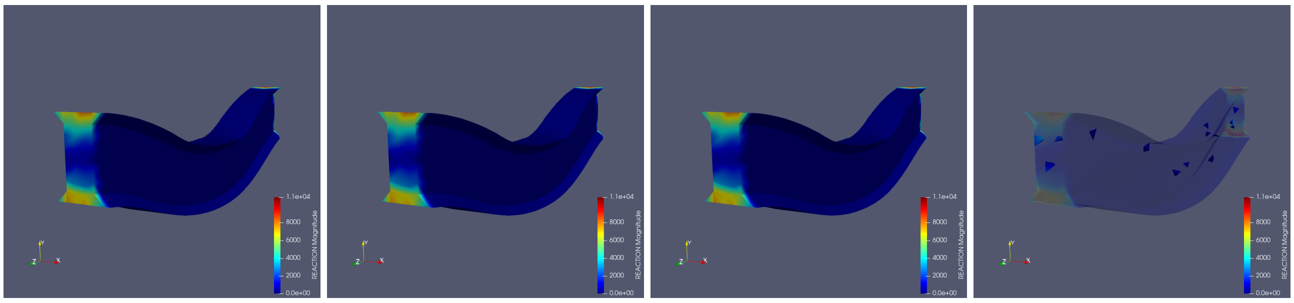
The FOM model was successfully hyper-reduced from 12460 elements to only 10. In figure 31 it is shown the *fixed surface 1's* reactions for the FOM, ROM, and HROM models, and in figure 32 the *fixed surface 2's* reactions for the FOM, ROM, and HROM models. The reduction was performed for two different tolerances to validate the results and errors, the L_2 errors are shown in tables 6 and 7 for the displacement and reaction field respectively.

Displacement			
Tolerance	Modes	L_2 Error ROM	L_2 Error HROM
1.00E-06	4	1.54E-06	1.33e-05
1.00E-08	7	2.91E-08	1.68E-06

Table 6: Rubber Fixed Ended Beam displacement field L_2 error for different truncated modes

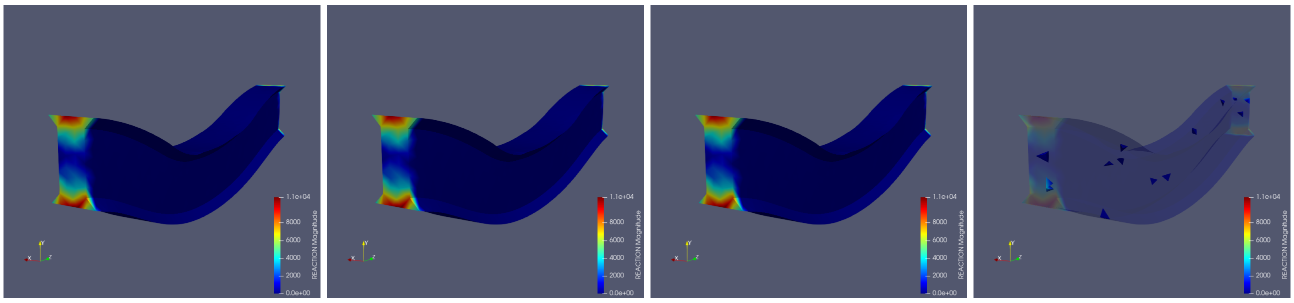
Stress			
Tolerance	Modes	L_2 Error ROM	L_2 Error HROM
1.00E-06	4	2.88E-06	1.36E-05
1.00E-08	6	3.91E-08	5.85E-06

Table 7: Rubber Fixed Ended Beam reaction field L_2 error for different truncated modes



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 31: FOM, ROM, and HROM Rubber Fixed Ended Beam Reactions, Surface 1



(a) FOM (b) ROM (c) HROM (d) HROM e_s

Figure 32: FOM, ROM, and HROM Rubber Fixed Ended Beam Reactions, Surface 2

PART V

APPLICATION

11 Mathworks

MathWorks is a private corporation that specializes in mathematical computing software. Its top products include Matlab and Simulink, which support simulation and data analysis.

- Matlab: Numerical computing system that offers an integrated development environment (IDE) with its own programming language (M language).
- Simulink: Visual programming environment, which works on top of the Matlab programming environment.
  - Simscape - Quickly model physical systems within the Simulink environment.

MATLAB®
&SIMULINK®

Figure 33: Matlab, Simulink, & Mathworks logo

Currently, Mathworks is capable of running multibody simulations successfully involving Rigid and Flexible Bodies for linear behaviour using Simscape's *Reduced Order Flexible Solid* block, which applies the *finite element import method* defined in Modeling Flexible Bodies with Simscape Multibody as an "*approximation of a flexible body as the superposition of a rigid-body model and a deformation model (their motions shown in figure 34). The rigid-body model captures the rotation and translation of the body as if it did not deform at all. The deformation model calculates the elastic deflections at selected points throughout the body as though the body had been pinned in place. The two models connect through separate components known as deflection joints*". The block needs the information of the *Mass, Stiffness, and Damping* matrices and the nodal location of the interface frames (surface connections to other Simscape Multibody elements) to perform the analysis.

11.1 Problem definition

Mathworks presents a test model that is an excavator arm as a flexible body as a benchmark problem for the flexible body Simscape's block (follow this link for more information Flexible Dipper Arm). The body is part of a multibody system (see figure 36), however this paper will focus more on the ROM block and its features.

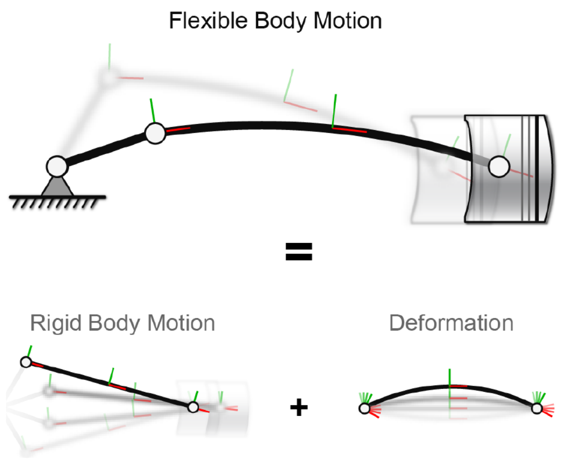


Figure 34: Flexible body motion as a superposition of rigid body motion and deformation (Image taken from Mathworks)

There are several ways to generate the reduced-order data required by the *Reduced Order Flexible Solid* block. Typically, you generate a substructure (or super-element) by using finite-element analysis (FEA) tools. *Mathworks* uses the "*Partial Differential Equation Toolbox*" to create a reduced-order model for a flexible dipper arm performing a *simplified Craig-Bampton* modal analysis which provides a "low fidelity" analysis of the matrices. Meanwhile Kratos with its fast solver settings, propose to obtain the stiffness matrix by performing a static condensation process to replace the stiffness matrix from the PDE toolbox as shown in figure 35, providing a "higher fidelity" model and the possibility to create a ROM and/or HROM to post-process the variables such as the displacement, reaction, and stress field of the flexible body. It is worth mentioning that the static condensation method was chosen to extend the possibility for a material and/or geometrical non-linear analysis.

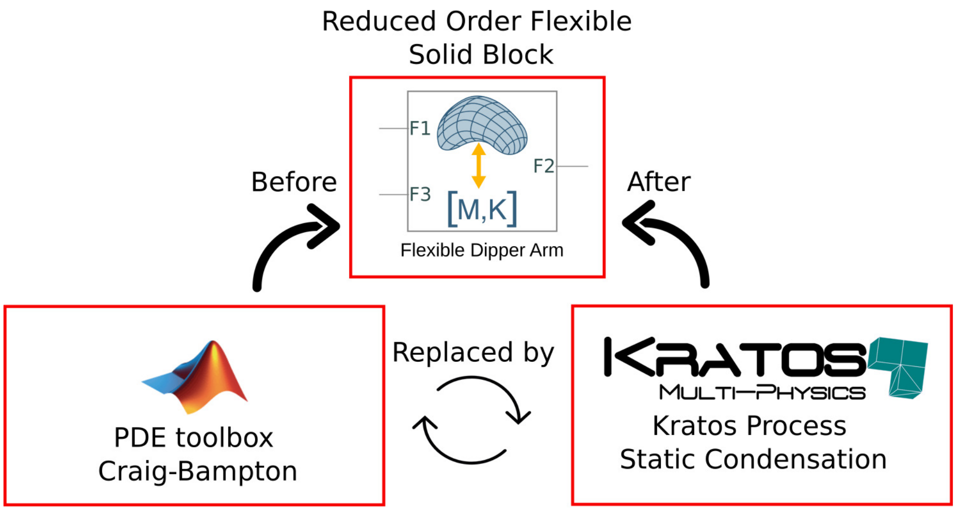


Figure 35: Reduced Order Flexible Solid Block Diagram

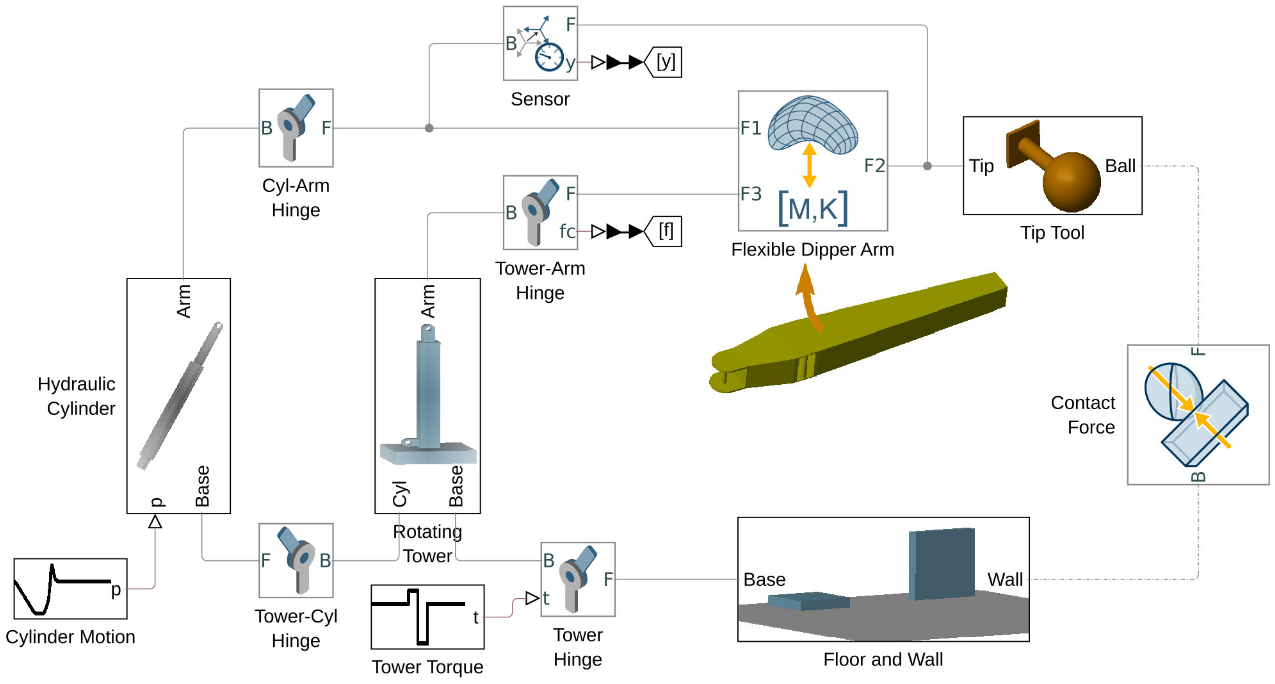
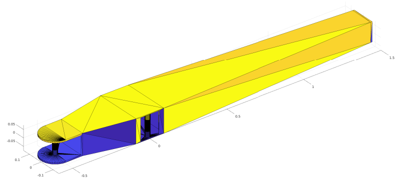


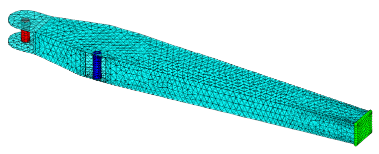
Figure 36: Simulink's Flexible Dipper Arm Multibody Problem

11.2 Kratos-Mathworks interface

To be able to interface the information coming from Kratos Multiphysics with Simulink, it is first necessary to generate the geometries and parameters for both software (*detailed information of how to build the Kratos' model can be found in the Appendix F*). In figure 37 and 38 the geometries of the different software for the solution of the proposed problem are shown.

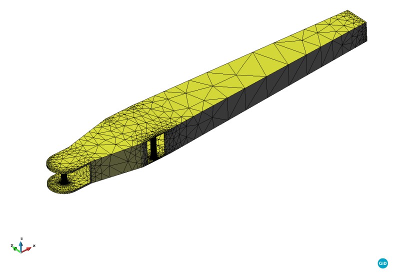


(a) Geometry Excavator Dipper Arm Mathworks



(b) Mesh and Interface Frames Excavator Dipper Arm Mathworks

Figure 37: Matlab's & Simulink's Geometry


(a) Geometry and Mesh Excavator Dipper Arm GiD

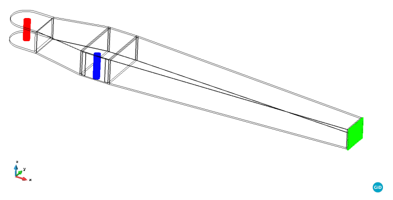

(b) Interface Frames Excavator Dipper Arm GiD

Figure 38: Kratos' Geometry

11.2.1 Kratos' static condensation process

An important step is to identify the interface frames of the flexible body (red, blue, and green) and assign a master node usually localized in the center of the surface. Kratos Multiphysics assigns the project parameters via a *json* file on which the *static condensation* process needs to be called in the following way:

```
"processes"          : {
    "constraints_process_list" : [{
        "python_module" : "static_condensation_process",
        "kratos_module" : "KratosMultiphysics.StructuralMechanicsApplication
            ",
        "Parameters"    : {
            "sub_model_part_list": ["Interface_frame_1","Interface_frame_2",
               "Interface_frame_3"],
            "list_of_master_coordinates": [[-0.5,0,0],[1.5,0,0],[0,-0.13,0]]
               ,
            "eps_perturbation": 1e-5
        }
    }],
},
```

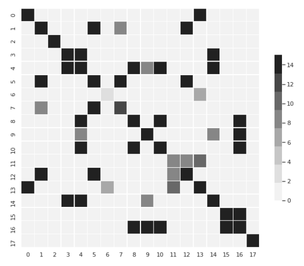
where the "sub_model_part_list" contains the names of the interface frames, the coordinates of the master nodes are contained in the "list_of_master_coordinates" and the "eps_perturbation" is a user parameter that controls the perturbation of the displacements/rotations that the model will be subjected during the static condensation process. This process as mentioned before, will output a *json* file with the reduced stiffness matrix (size of 18x18 on this case) and the coordinates of the master nodes.

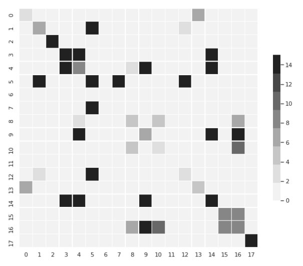
To validate the results of the static condensation process, the flexible dipper arm was reduced for different meshes with the *Craig-Bampton PDE toolbox's method* without considering any internal modes and with the static condensation process from Kratos. To show the convergence, a color map of the 18x18 matrices are shown in figure 39, highlighting that as the mesh is refined, the error decays, and specially in the diagonal terms.

11.2.2 Matlab interface

There are two ways of running the Kratos' Python file from Matlab's work-space. The first an easier option for a unique call is:

```
system("python MainKratos.py")
```


(a) 2800 Nodes


(b) 33000 Nodes

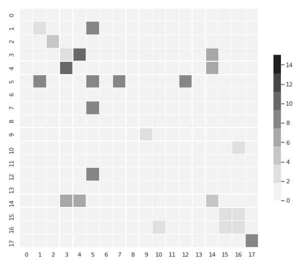

(c) 118000 Nodes

Figure 39: Static Condensation Convergence Color Map

And for specific modules inside the “MainKratos.py” file:

```matlab
pathToKratos = fileparts(which("MainKratos.py"));
if count(py.sys.path,pathToKratos) == 0
    insert(py.sys.path,int32(0),pathToKratos);% Add python module ...
                                               % to Matlab
end
py.MainKratos.RunKratos();
```

For a linear small-displacement problem, it is only needed to provide a constant reduced order stiffness matrix for the whole simulation, therefore, with the help of Matlab's work-space interface with Simulink's, the Kratos' stiffness matrix can be read as a .mat file and directly assign it to the Simulink's works-space by the following commands:

```matlab
fname = 'Stiffness_Matrix.json'; % Read stiffness matrix json file ...
                                 % created by Kratos' process
K = jsondecode(fileread(fname)); % Save stiffness matrix on ...
                                 % current workspace
arm.K = K.StiffnessMatrix; % Replace Simscape's stiffness matrix ...
                           % with Kratos'

%% Modify Simscape model
model_name = "flexible_dipper_arm_kratos"; % Name of simulink project
mdlWks = get_param(model_name,'ModelWorkspace'); % Save Simscape's ...
                                                 % Model Workspace
mdlWks.setVariablePart('arm.K',arm.K); % Replace Stiffness Matrix
mdlWks.setVariablePart('arm.P',arm.P); % Replace Interface Frame ...
                                       % Origins (master nodes
                                          coordinates)
```

Once the Kratos' reduced stiffness matrix is replaced on the Simscape block, the Simulink simulation can be run as:

```matlab
%% Run Simscape model with new parameters
sim(model_name);
```

11.2.3 Simulink results

The results coming from Simulink can be transferred to Matlab's work-space (see figure 40) to show the displacements and rotations of the master nodes.

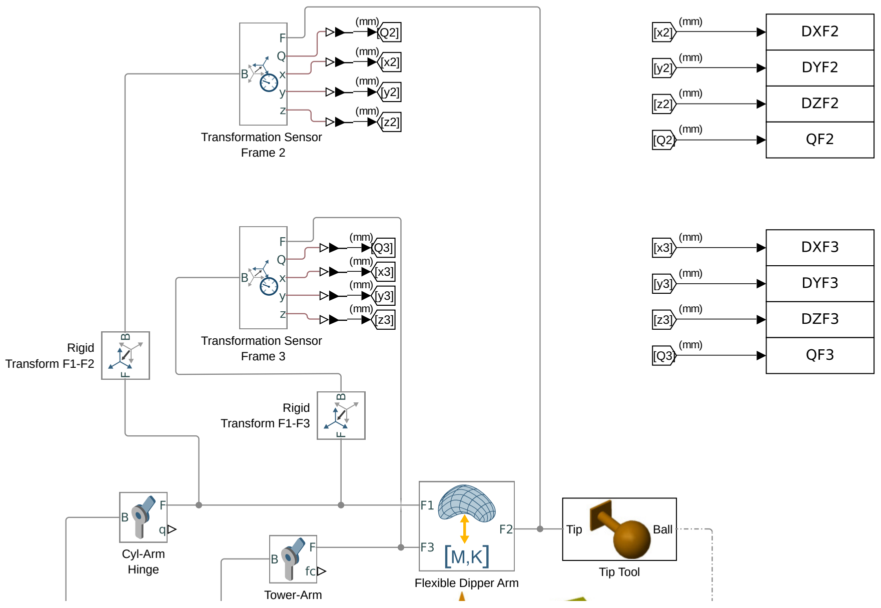


Figure 40: Simulink's results to Matlab's work-space

In this case, the displacements and rotations (quaternions) of the Interface Frame 2 and 3 are obtained with respect to the reference Interface Frame 1 to denoise rigid body motion. This transformation can be set in Matlab's work-space as follows:

```matlab
%% Set the translation of the Rigid Transform blocks appropriately
P12 = arm.P(1,:) - arm.P(2,:); % Compute the relative offset ...
                               % between the interface frames ...
P13 = arm.P(1,:) - arm.P(3,:); % and the common reference frame

% Use the following command to set the value of the Cartesian offset
% translation
set_param(model_name+'/Rigid Transform F1-F2','
    TranslationCartesianOffset', ['[' num2str(P12) ']']);
set_param(model_name+'/Rigid Transform F1-F3','
    TranslationCartesianOffset', ['[' num2str(P13) ']']);
```

The results can be plotted in graphs (see figure 41) and later saved as a Matlab structure in a .mat file ("sim_res.mat"). A proposed structure of the displacement's results is shown in the appendix G.

11.2.4 Results

Simulink provides many ways to visualize the results, via Simulink's scope (Graphs and Tables) and as mentioned before, the results can be exported to Matlab's work-space and simply plot them as in figure 41.

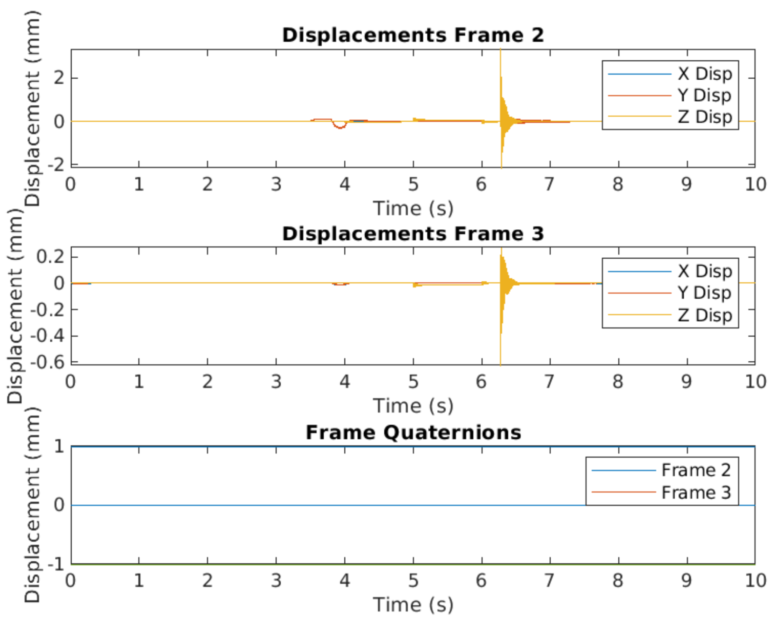
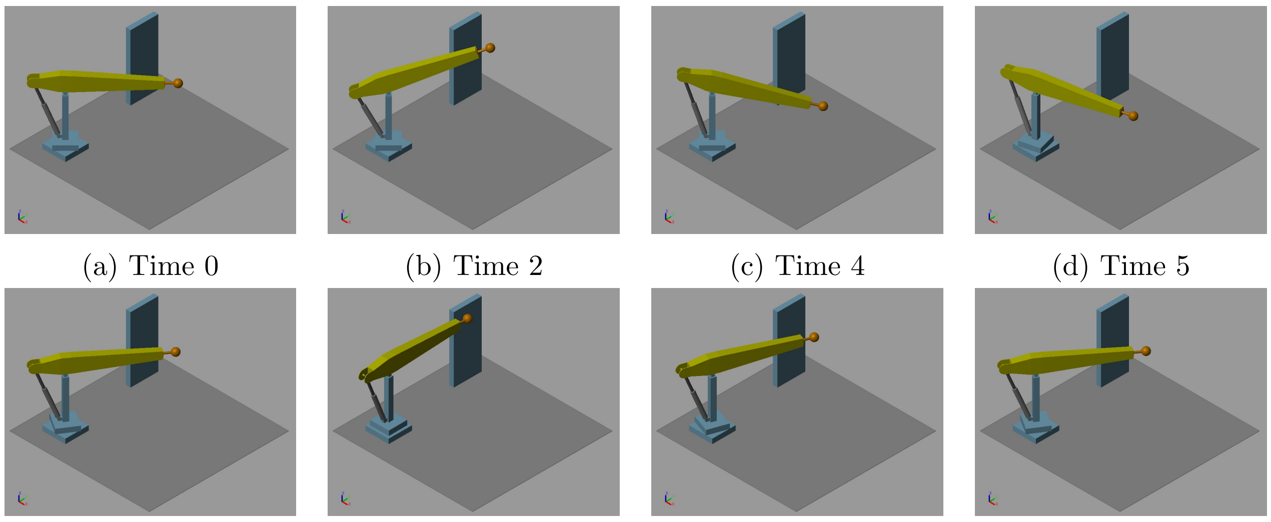


Figure 41: Displacements and rotations of interface frames 2 and 3 with respect to reference frame 1

When the multibody/control problem is solved, Simulink offers a visual demo of the dynamic motion that you can export as a collection of images (see figure 42) and/or a video.



(a) Time 0 (b) Time 2 (c) Time 4 (d) Time 5

(e) Time 6 (f) Time 6.27 (g) Time 8 (h) Time 10

Figure 42: Flexible Dipper Arm Dynamic Motion

However, Mathworks does not let the user get a post-process of the displacement, stress, and/or reaction field.

11.3 Post-process in Kratos

For a linear problem undergoing small displacements, the training stage for obtaining an static condensation works perfectly to train an HROM. This implies that Kratos can offer a fast post-processing of the displacement, stress, and reaction field.

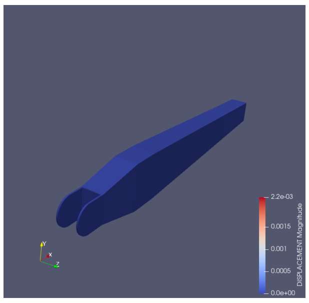
As proposed in appendix G, the results of the displacements and rotations can be saved in a .mat file as an structure that later can be exported to Kratos, and with the help of a proposed process "*simulink_postprocessing_process*" the boundary conditions on the master nodes are applied for an specific time-step range (to avoid lack of memory for large simulations). The additional steps to post-process the results are:

- Add "*rom_basis_process*" to project parameters: Saves the Snapshot Matrices for the displacement, stress, and/or reaction field and creates the modal basis for a given tolerance to reduce the flexible body solid.
- Add "*Empirical Cubature*" Kratos' strategy to *Main_Kratos.py*: Creates the hyper-reduced model part with its respective weights.
- Create an HROM project parameters and include the "*simulink_postprocessing_process*": Applies the boundary conditions to the master nodes for an specific time-step range and outputs the post-processing results.

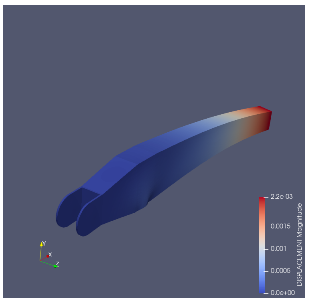
The parameters for the "*simulink_postprocessing_process*" are:

```
"processes"         : {
    "constraints_process_list" : [{
        "python_module" : "simulink_postprocessing_process",
        "kratos_module" : "KratosMultiphysics.RomApplication",
        "Parameters"    : {
            "model_part_list": ["Interface_frame_1","Interface_frame_2","Interface_frame_3"],
            "list_of_master_coordinates": [[-0.5,0,0],[1.5,0,0],[0,-0.13,0]],
            "time_step_range": [start_time_step,end_time_step]
        }
    }]
}
```

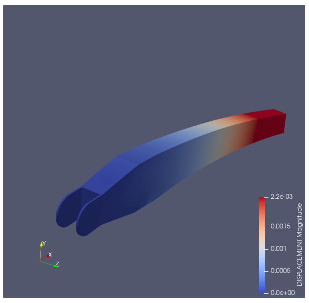
The critical point of this simulation comes around time 6.27, where the *Tip Ball* hits the wall producing a contact force and a deformation on the flexible body(contact in figure 42f). The corresponding time-step range to capture moments before and after the contact force are from 5200 to 7200 (6.2585 to 6.3338 seconds). In figures 43, 44, and 45 the displacement, stress, and reaction field are presented respectively for different times to show the oscillation of the dipper arm as it hits the wall.



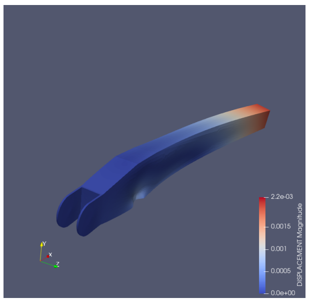
(a) Time 6.2585



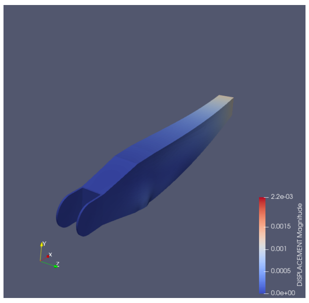
(b) Time 6.2737



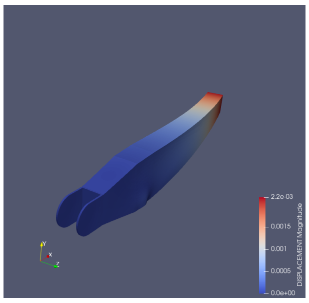
(c) Time 6.2745



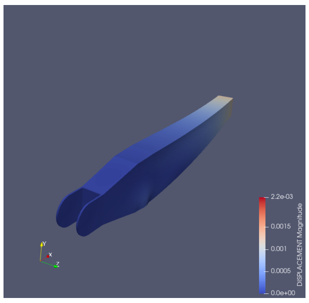
(d) Time 6.2763



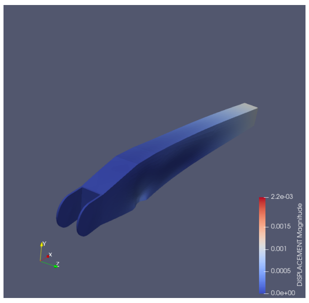
(e) Time 6.2802



(f) Time 6.2814

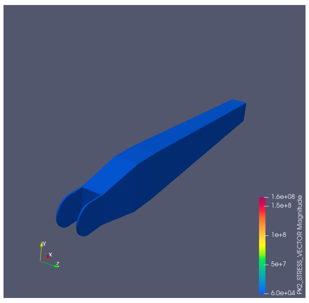


(g) Time 6.2824

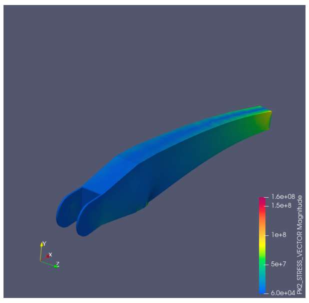


(h) Time 6.2846

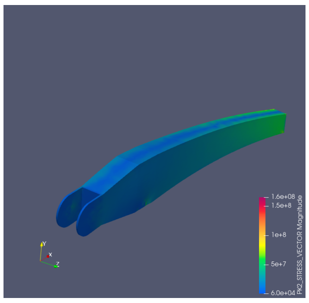
Figure 43: Flexible Dipper Arm Displacement Field (x100) Post-process



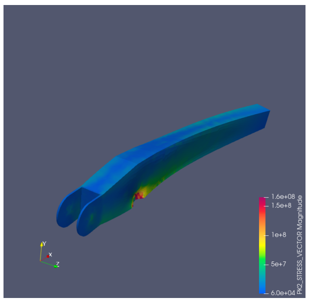
(a) Time 6.2585



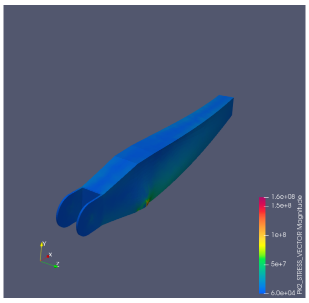
(b) Time 6.2737



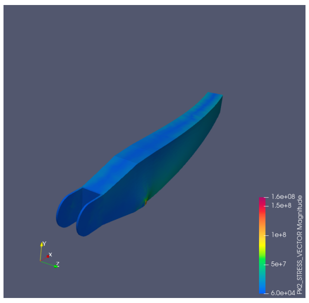
(c) Time 6.2745



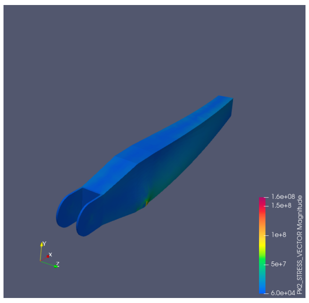
(d) Time 6.2763



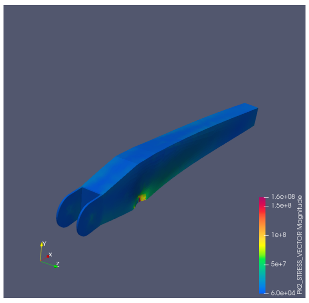
(e) Time 6.2802



(f) Time 6.2814

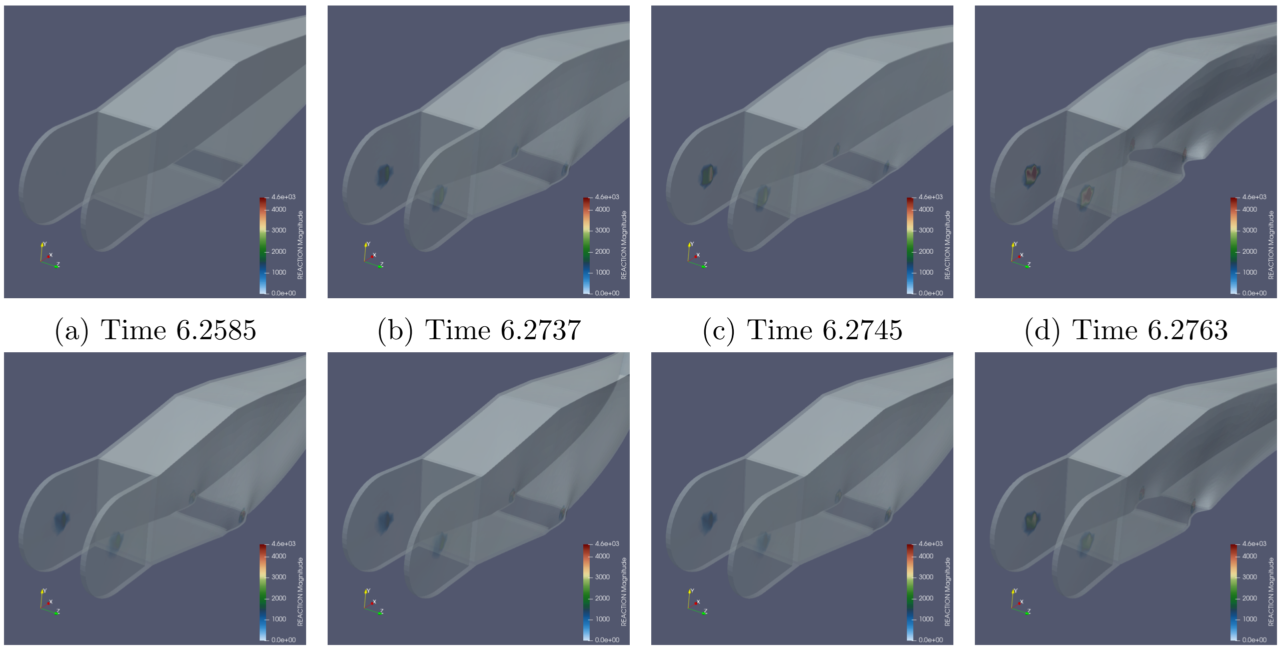


(g) Time 6.2824



(h) Time 6.2846

Figure 44: Flexible Dipper Arm Stress Field (x100) Post-process



(a) Time 6.2585 (b) Time 6.2737 (c) Time 6.2745 (d) Time 6.2763

(e) Time 6.2802 (f) Time 6.2814 (g) Time 6.2824 (h) Time 6.2846

Figure 45: Flexible Dipper Arm Reaction Field (x100) Post-process

PART VI

CONCLUSIONS AND FUTURE WORK

12 Conclusions

Reduced order modeling techniques has been growing lately due to its multiple uses, such as the solution of multibody/control problems, augmented and virtual reality, among many others applications. The main objective of this paper was the reconstruction of the stress and reaction field, which was successfully carried out via a combination of the proper orthogonal decomposition 4.1 and the gappy data reconstruction methodology 4.3. The errors obtained for the linear and non-linear analysis fulfilled the expected behaviour, being of similar order to its corresponding basis tolerance. The validation of the results was carried out via 3D FOM, ROM, and HROM linear and non-linear simulations with complex geometries, offering a visual representation together with an error convergence analysis.

The methodology employed for the reconstruction of the stress and reaction field is completely extrusive, unlike the already implemented for the displacement field. This means that the reaction and stress field doesn't have any impact on the solution of the ROM and HROM, the construction of these basis will only impact their own skin reconstruction (stress and reaction). Being completely extrusive forces any other reconstructed field to be displacement dependent, on the other hand, it has an strong advantage on reconstructing any field with similar methodology as the ones proposed, yielding an easy implementation.

Solving control and/or multibody problems represents a great engineering challenge, both due to its complexity and the required memory size. Fortunately, Mathworks offers the ability to design and simulate highly demanding systems, combining control tools with rigid and/or flexible motion analysis blocks. However, as previously mentioned, flexible body blocks present two counterpoints:

- The results from the PDE toolbox (simplified Craig-Bampton) analysis are low fidelity.
- It does not allow post-processing of displacements, forces, and reactions of the structural elements.

Having a low fidelity analysis for the flexible block can have a great impact on the final behavior of the control and/or multibody system since it has a direct dependence on all its connections. In addition, the analysis of these systems regularly goes hand in hand with the structural design of their components, so by not offering post-processing of deformations, stresses, and reactions, it will not fulfill the analysis needs.

Kratos Multiphysics, is an extremely powerful and precise tool, allowing through its static condensation process the reduction of a stiffness matrix with higher fidelity than the obtained with Mathworks' PDE toolbox (simplified Craig-Bampton analysis of the method explained in section 4.4.1). It also allows using the condensation process as the training base for a small-displacements linear problem (i.e. flexible dipper arm) to build an HROM that properly and quickly allows the user to carry out detailed post-processing (displacement, stress, and reaction field) of a given range of critical time-steps for the design of structural elements.

Although the static condensation process allows obtaining the stiffness matrix, it is necessary to add the possibility of obtaining a mass matrix with a modal analysis technique similar to Craig-Bampton's. The decision to carry out the static condensation process is to extend the possibility of carrying out a nonlinear geometric and/or material analysis in the future.

13 Future work

13.1 HROM

The successful extrusive reconstruction of the stress and reaction field, potentially indicates that most of the variables coming from the FOM analysis could be hyper-reduced, reconstructed, and projected onto the skin (e.g. internal variables). Future work will be to reconstruct the most important variables for the structural mechanics problem, followed by an expansion to the fluids mechanics field.

One counterpoint of the applied methodology when reducing the models is that the user has to define a training space good enough to succeed, being time-consuming and troublesome. Therefore, a personal goal is to perform a Kratos' process were given a training space, the process would be capable to decide if its well defined or not by defining a test space similar to the training space and add it whenever a threshold on the L_2 error comparison is over-passed, enriching the training the space.

13.2 Multibody systems

The next step to enrich the control/multibody Mathworks' analysis is to provide an interface process and surrogate model with Kratos Multiphysics that enables the geometric and material non-linearity of the structural elements. The main difficulty towards this implementation is building the *data exchange interface* that lets an efficient communication between both software during run-time. Once this is established, the second challenge is to create an optimal threshold inside either Kratos or Mathworks that indicates the need of updating the flexible body's stiffness matrix every time is needed throughout the control/multibody nonlinear analysis.

13.2.1 Methodology

The idea of building a run-time *data exchange interface* between Kratos and Mathworks can be achieved with the help of MATLAB function blocks inside Simulink work-space (see figure 46), these blocks "*generate binary code or C/C++ MATLAB executable (MEX) code from the block and integrates this code with the model. The MATLAB Function block uses the same infrastructure as MATLAB Coder, which you use to generate C/C++ code from MATLAB code outside of Simulink*" (visit MATLAB Function Blocks for more information). This has some limitations to the Matlab functions you can call, therefore, it is needed to create a Wrapper that enables a Matlab function to explode all capabilities. A proposed block and corresponding wrapper are shown bellow:

Matlab Function Block

```matlab
function [time_step, aux]= myBlock(time_step,DXF2,DYF2,DZF2,QF2,DXF3,
    DYF3,DZF3,QF3)
coder.extrinsic('myWrapper')
threshold_flag = myWrapper(time_step,DXF2,DYF2,DZF2,QF2,DXF3,DYF3,DZF3
    ,QF3);
```

Matlab Function Wrapper

```matlab
function threshold_flag = myWrapper(varargin)
%% Run Kratos HROM
threshold_flag=py.MainKratos_HROM.RunKratosHROM(varargin{:});
%% Run Static Condensation for Non-linear analysis
if threshold_flag==1
    %%Stop simulation and save the dynamic states
```

Simulink's solver doesn't let the user modify the physical parameters simulation during run-time to avoid instabilities, therefore it is necessary to stop the simulation and save the final dynamic state. Once the simulation is stopped, the new nonlinear static condensation Kratos' process can be executed and the Simscape's block parameters can be updated as usual. The simulation can be restarted from the last saved dynamic state with the new modified parameters until a new threshold is over-passed. The techniques to stop the simulation, save the dynamic states with its corresponding useful variables is still on progress, only the main idea of the workflow was explained before, in addition, the nonlinear static condensation process methodology follows the steps:

1. Save the last master node's displacements and rotations into a .mat structure file and read them inside Kratos.

2. Set the displacements and rotations of the master node's as initial condition to the HROM model in Kratos.

3. Apply the static condensation process to the deformed state to obtain the new reduced stiffness matrix.

4. Save the reduced stiffness matrix into a json file.

The implementation of this process in conjunction with the data exchange interface will allow the user to carry out nonlinear geometric and material analysis of the structural components of a control/multibody system under Simulink's solver.

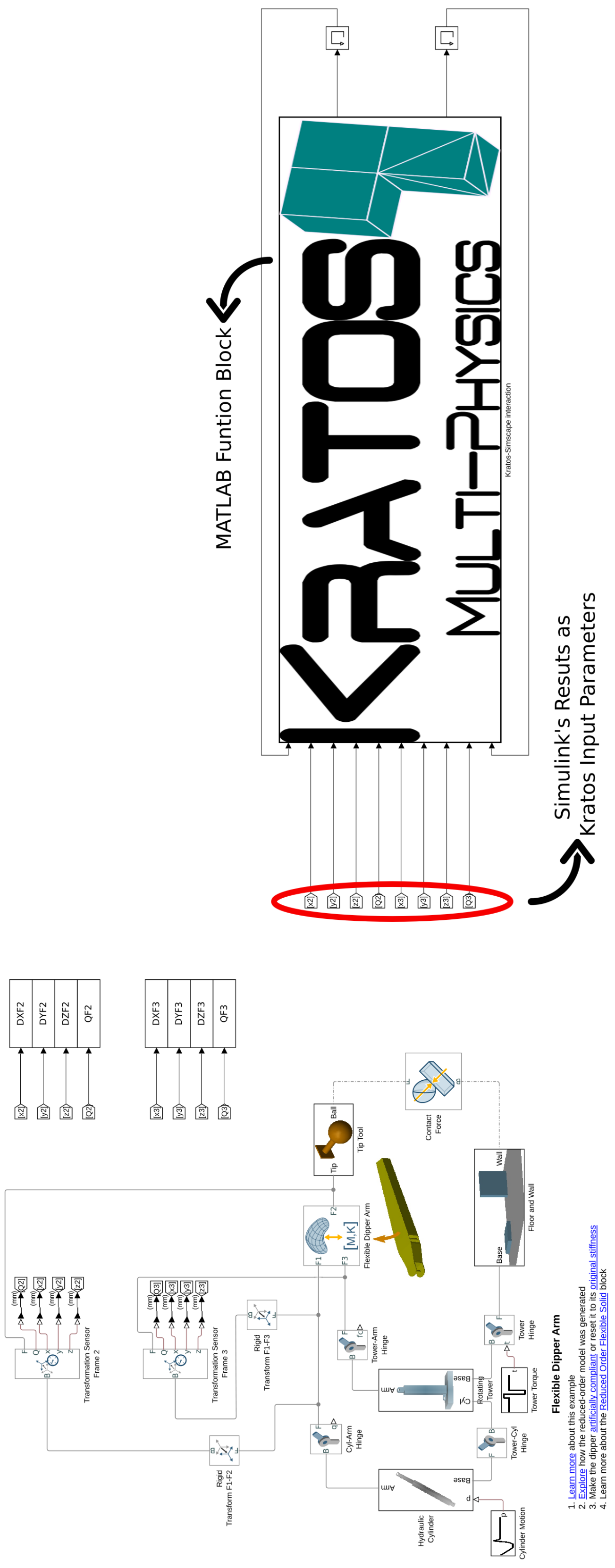

MATLAB Funtion Block
Simulink's Resuts as Kratos Input Parameters
KRATOS MULTI-PHYSICS
Kratos-Simscape interaction
DXF2
DYF2
DZF2
QF2
DXF3
DYF3
DZF3
QF3
Transformation Sensor Frame 2
Transformation Sensor Frame 3
Rigid Transform F1-F2
Rigid Transform F1-F3
Flexible Dipper Arm
Tip Tool
Contact Force
Floor and Wall
Tower-Arm Hinge
Cyl-Arm Hinge
Rotating Tower
Tower Hinge
Tower Torque
Tower-Cyl Hinge
Hydraulic Cylinder
Cylinder Motion
Flexible Dipper Arm
1. Learn more about this example
2. Explore how the reduced-order model was generated
3. Make the dipper artificially compliant or reset it to its original stiffness
4. Learn more about the Reduced Order Flexible Solid block

Figure 46: Flexible Dipper Arm Simulink-Kratos Interface

References

[1] S. L. Brunton and J. N. Kutz, *Data-Driven Science and Engineering*, 2019.

[2] P. Flores, “Fundamental concepts in multibody dynamics,” *SpringerBriefs in Applied Sciences and Technology*, vol. 168, 2015.

[3] C. Felippa, “Multifreedom constraints i & ii.” *Introduction to Finite Element Methods*, 2014.

[4] C. Eckart and G. Young, “The approximation of one matrix by another of lower rank,” *Psychometrika*, vol. 1, 1936.

[5] N. Halko, P. G. Martinsson, and J. A. Tropp, “Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions,” *SIAM Review*, vol. 53, 2011.

[6] R. Penrose, “A generalized inverse for matrices,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 51, 1955.

[7] R. Macausland, “The moore-penrose inverse and least squares,” *Advanced Topics in Linear Algebra*, 2014.

[8] E. Oñate, *Structural Analysis with the Finite Element Method*, 2009, vol. 1.

[9] A. Peiret, F. González, J. Kövecses, and M. Teichmann, “Multibody system dynamics interface modelling for stable multirate co-simulation of multiphysics systems,” *Mechanism and Machine Theory*, vol. 127, 2018.

[10] R. Reyes and R. Codina, “Projection-based reduced order models for flow problems: A variational multiscale approach,” *Computer Methods in Applied Mechanics and Engineering*, vol. 363, 2020.

[11] J. A. Hernández, M. A. Caicedo, and A. Ferrer, “Dimensional hyper-reduction of nonlinear finite element models via empirical cubature,” *Computer Methods in Applied Mechanics and Engineering*, vol. 313, 2017.

[12] J. A. Hernández, “A multiscale method for periodic structures using domain decomposition and ecm-hyperreduction,” *Computer Methods in Applied Mechanics and Engineering*, vol. 368, 2020.

[13] R. Everson and L. Sirovich, “Karhunen–loève procedure for gappy data,” *Journal of the Optical Society of America A*, vol. 12, 1995.

[14] R. J. Guyan, “Reduction of stiffness and mass matrices,” *AIAA Journal*, vol. 3, 1965.

[15] W. B. Haile, “Primer on the craig-bampton method,” *Transformation*, 2000.

[16] A. A. Shabana, *Dynamics of multibody systems*, 2013, vol. 9781107042650.

[17] M. Vaz, P. A. Muñoz-Rojas, and G. Filippini, “On the accuracy of nodal stress computation in plane elasticity using finite volumes and finite elements,” *Computers and Structures*, vol. 87, 2009.

Appendix

A Rom Basis Process

To create the modal basis for the different reconstructed fields, "*rom_basis_process*" needs to be added to the project parameters' processes list as follows:

```
"processes"         : {
    "list_other_processes"     : [{
        "python_module" : "rom_basis_process",
        "kratos_module" : "KratosMultiphysics.RomApplication",
        "Parameters"    : {
            "variables_to_create_basis"     : ["DISPLACEMENT","REACTION", "
               PK2_STRESS_VECTOR"],
            "displacement_basis_tolerance" : 1e-8,
            "reaction_basis_tolerance" : 1e-8,
            "stress_basis_tolerance": 1e-6,
            "restricted_model_part": "name_of_restricted_model_part"
        }
    }
    ]
}
```

B Empirical Cubature Strategy

To select the elements that govern the HROM, the Rom Application's *Empirical Cubature Strategy* needs to be called in python after creating the ROM basis as follows:

```
""
#####################################################
#                   TRAIN HROM                      #
#####################################################
with open("ProjectParameters.json",'r') as parameter_file:
    parameters= KratosMultiphysics.Parameters(parameter_file.read())
model= KratosMultiphysics.Model()
simulation= StructuralMechanicsAnalysisROM(model,parameters,"EmpiricalCubature")
simulation.Run()
```

This will create a *ElementsAndWeights.json* and *Hyper_Reduced_Model_Part.mdpa* files.

C HROM Simulation

To run an HROM simulation, it is needed to create the project parameters files that apply the boundary conditions and parameters to the *Hyper Reduced Model Part*, and that sets the corresponding weights to the HROM elements coming from the *ElementsAndWeights.json* file as follows:

```
""
class RunHROM(StructuralMechanicsAnalysisROM):

    def ModifyInitialGeometry(self):
        """Here is the place where the HROM_WEIGHTS are assigned to the selected
    elements and conditions"""
        super().ModifyInitialGeometry()
        computing_model_part = self._solver.GetComputingModelPart()
        ## Adding the weights to the corresponding elements
```

```python
        with open('ElementsAndWeights.json') as f:
            HR_data = json.load(f)
            for key in HR_data["Elements"].keys():
                computing_model_part.GetElement(int(key)+1).SetValue(romapp.HROM_WEIGHT, HR_data["Elements"][key])
            for key in HR_data["Conditions"].keys():
                computing_model_part.GetCondition(int(key)+1).SetValue(romapp.HROM_WEIGHT, HR_data["Conditions"][key])

###################################################
#                   RUN HROM                      #
###################################################
with open("ProjectParameters_HROM.json",'r') as parameter_file:
    parameters = KratosMultiphysics.Parameters(parameter_file.read())
model = KratosMultiphysics.Model()
simulation = RunHROM(model,parameters)
simulation.Run()
```

D Stress Reconstruction Process

To visualize the reconstruction of the stress field on the skin of the HROM, the module "*stress_reconstruction_process*" needs to be added to the HROM project parameters' processes list as follows:

```
"processes"        : {
    "list_other_processes"     : [{
        "python_module" : "stress_reconstruction_process",
        "kratos_module" : "KratosMultiphysics.RomApplication",
        "Parameters"    : {
            "variable_to_reconstruct" : "PK2_STRESS_VECTOR"
        }
    }]
}
```

E Reaction Reconstruction Process

To visualize the reconstruction of the reaction field on the skin of the HROM, the module "*reaction_reconstruction_process*" needs to be added to the HROM project parameters' processes list as follows:

```
"processes"        : {
    "list_other_processes"     : [{
        "python_module" : "reaction_reconstruction_process",
        "kratos_module" : "KratosMultiphysics.RomApplication",
        "Parameters"    : {
            "variable_to_reconstruct" : "REACTION",
            "restricted_model_part" : "name_of_restricted_model_part"
        }
    }]
}
```

F Static condensation process

A simple geometry of a beam-like element is proposed with two surfaces that will be the interface frames to be connected to another Simscape Multibody elements model, such as joints, constraints, forces, and sensors. Each interface surface on the 3D model corresponds to a master node that contributes six degrees of freedom to a reduced-order model (ROM). Therefore, the need arises to implement a process in Kratos Multiphysics that allows to obtain the stiffness matrix of multiple master nodes that are part of a structural element in 3 dimensions.

F.1 Problem type

To create the model part, the GiD pre-process and post-process interface is used, which will allow to create a file with the extension **.mdpa** which contains the geometric information of the Model Parts and Sub-model Parts mesh.

To set an Structural Mechanics problem it is first needed to load the Kratos GiD GUI. This is done by doing the following sequence of commands **Data** → **Problem type** → **Kratos** in the top toolbar. Then the Kratos application market will appear in where the Structural application must be selected.

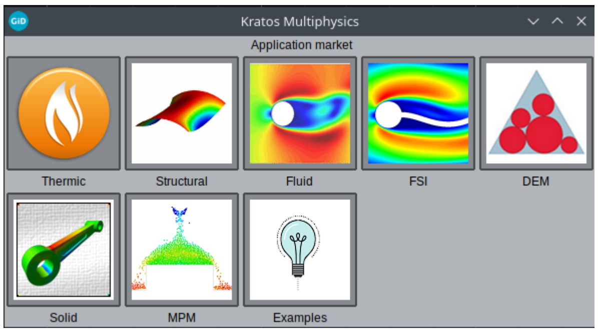


Figure 47: Application Market

After this, the structural mechanics dimension selection will appear. Among 2D or 3D dimensions, select the 3D structural mechanics button.

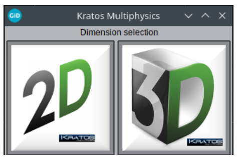


Figure 48: Dimension selection

F.1.1 Define the Geometry and Material Properties of the Beam-like element

First, the geometry of the beam-like element that will be the Model Part must be defined:

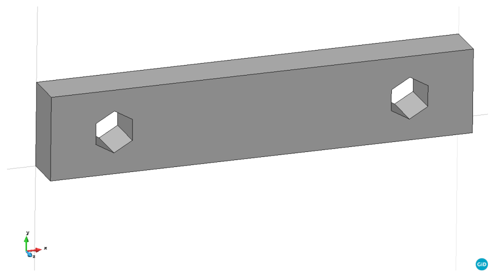


Figure 49: Beam-like Element Geometry

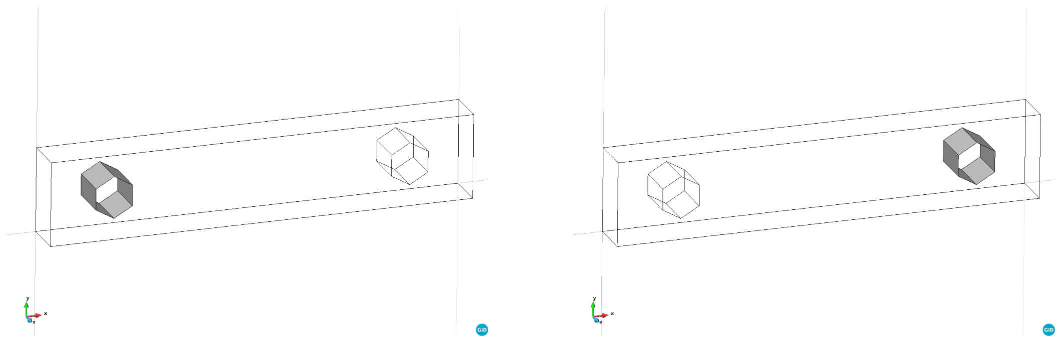
This solid was assigned with the properties listed below:

- Element: Solid small displacement
- Constitutive law: Linear Elastic
- Material: Steel
- Density: 7850 $\frac{kg}{m^2}$
- Young Modulus: 206.9e9 Pa
- Poisson Ratio: 0.29

This will generate a file with the extension "Materials.json" that has the parameters of the materials to be modeled.

F.1.2 Interface Frames

Once the properties of the Model Part have been assigned, the interface surfaces must be defined as sub-model parts in order for GiD to include them in the ".mdpa" file:



(a) Interface frame 1 (b) Interface frame 2

F.1.3 Master node coordinates

It is worth mentioning that the coordinates of the master node are normally considered in the respective center of the interface surfaces, however the process will allow to indicate any coordinate as the master coordinate, either a coordinate of an existing node, or a node outside of the model, as it is shown in the following figure:

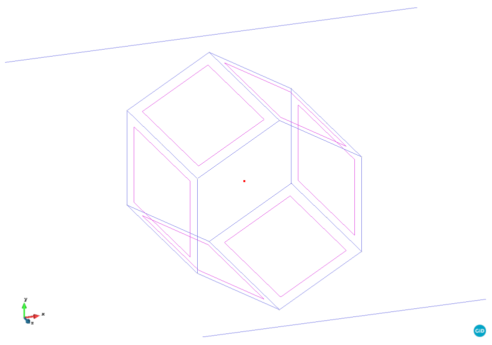


Figure 51: Master node 2

F.1.4 Create the finite element mesh

Finally, it is necessary to create the mesh that you want to use for modeling:

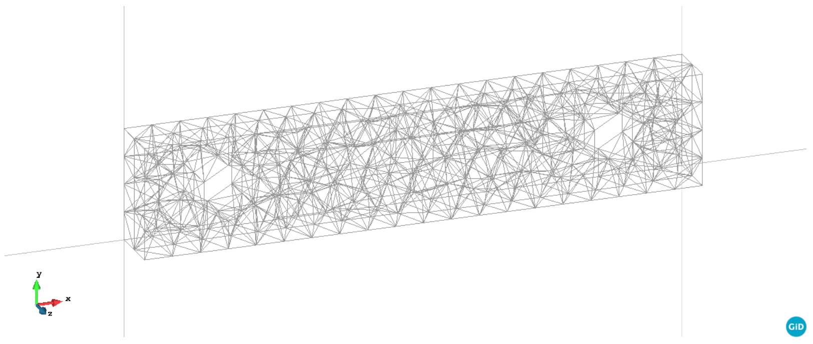


Figure 52: Mesh

F.1.5 Output files “.mdpa” and “.json”

Once the previous steps have been carried, GiD will generate the following files in the folder where this model has been saved:

- ***.mdpa**: File containing the mesh geometry.
- **ProjectParameters.json**: File containing the problem settings and BCs.
- **MainKratos.py**: Main script to run the simulation.
- ***Materials.json**: File containing the material properties.

F.2 Kratos Multiphysics

F.2.1 MainKratos.py

The main file of a Kratos simulation is a python script. It is responsible to load the required Kratos applications and to call the main Kratos functionalities as desired by the user.

```python
""
import KratosMultiphysics
from KratosMultiphysics.StructuralMechanicsApplication.
    structural_mechanics_analysis import StructuralMechanicsAnalysis
```

```python

if __name__ == "__main__":

    with open("ProjectParameters.json",'r') as parameter_file:
        parameters = KratosMultiphysics.Parameters(parameter_file.read())

    model = KratosMultiphysics.Model()
    simulation = StructuralMechanicsAnalysis(model, parameters)
    simulation.Run()
```

In the first lines, Kratos and the structural analysis are imported. Then the settings are read from the .json file and used to create an object of the structural analysis. In the last line, the structural simulation is executed.

F.2.2 ProjectParameters.json

The settings for a Kratos simulation are stored in a .json file. JSON is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. Kratos uses a thin wrapper around this syntax, the Parameters object.

The structure of this parameters file is the following:

- **problem_data**: General settings for the Kratos run
- **solver_settings**: Settings for the solvers, like analysis type, linear solver, etc.
- **processes**: Processes to apply.
- **output_processes**: Settings for the output.

F.2.3 Solver settings

Inside the solver settings, define the material properties and the model part by calling the respective Materials.json and ProjectName.mdpa file in the following way:

```json
"solver_settings"  : {
    "model_import_settings"               : {
                "input_type"      : "mdpa",
                "input_filename" : "ProjectName"
    "material_import_settings"            : {
                "materials_filename"  : "Materials.json"
    }
},
```

F.2.4 Processes

To run process **"static_condensation_process"**, it is needed to give the respective parameters inside the PorjectParameters.json file.
The process is constructed by providing a "Parameters" object, initialized by a string in json format. The input parameters of this process are simply the following:

- List of interface frames ("sub_model_part_list").
- Coordinates of master nodes of the interface frames ("list_of_master_coordinates").

- Perturbation parameter for infinitesimal translational displacement (“eps_perturbation”).

Here is an example of how the parameters should be delivered:

```
"processes"          : {
    "constraints_process_list" : [{
        "python_module" : "static_condensation_process",
        "kratos_module" : "KratosMultiphysics.StructuralMechanicsApplication",
        "Parameters"    : {
            "sub_model_part_list": ["Interface_frame_1","Interface_frame_2"],
            "list_of_master_coordinates": [[0.15,0.1,0.025],[0.55,0.1,0.025]],
            "eps_perturbation": 1e-5
        }
    }],
},
```

F.2.5 Run Simulation

Once the model parameters have been correctly defined, the simulation is ready to run. For this, just call the MainKratos.py file from the console:

```
$ python MainKratos.py
```

The result of the process is written in “*json*” format and can be found as “Master_Stiffness.json”.

G Simulink’s Results Structure

```
%% Export data to Kratos
sim_res.Time = DXF2.time; % Obtain time steps array
sim_res.Num_of_frames = length(arm.P);
sim_res.Kratos_Interface_Frame_Origins = K.InterfaceFrameOrigins;
sim_res.Kratos_Displacement = K.Displacement; % Displacement of Kratos' process
sim_res.Kratos_Rotation = K.Rotation; % Rotation of Kratos' process in radians
for i=2:sim_res.Num_of_frames
    temp_name = strcat( 'Interface_',num2str(i)); % Dynamic Field-Names
    sim_res.(temp_name).Disp_x = eval(strcat('DXF',num2str(i))).data; % Obtain displacements on x array
    sim_res.(temp_name).Disp_y = eval(strcat('DYF',num2str(i))).data; % Obtain displacements on y array
    sim_res.(temp_name).Disp_z = eval(strcat('DZF',num2str(i))).data; % Obtain displacements on z array
    sim_res.(temp_name).Quaternion = eval(strcat('QF',num2str(i))).data; % Obtain quaternions array
end
save('sim_res.mat','-struct','sim_res') % Create a .mat file with the results structure
```