POLITECNICO DI MILANO

SCHOOL OF CIVIL, ENVIRONMENTAL AND LAND MANAGEMENT ENGINEERING

MASTER OF SCIENCE IN CIVIL ENGINEERING - STRUCTURES

# MODEL ORDER REDUCTION

# H-REFINEMENT

Supervisor: Dr. Giorgio NOVATI[a]

Co-Supervisor: Dr. Angelo SIMONE[b]

Candidate:

Aleksei SUVOROV[a]

836339

Politecnico di Milano[a], Delft University of Technology[b]

Academic Year 2016/2017

## CONTENTS

1 Abstract ..... 5

Italian abstract ..... 6

2 Introduction ..... 7

3 Problem Statement ..... 9

3.1 Strain-softening Perzyna Viscoplasticity ..... 11

3.2 Inviscid Burgers Equation ..... 12

4 Methods ..... 17

4.1 Full Order Model ..... 17

4.2 Data Collection ..... 19

4.3 Data Normalization ..... 20

4.4 Tree Construction ..... 21

4.4.1 Clustering ..... 21

4.4.2 Clustering typology ..... 22

4.4.3 k-means clustering ..... 25

4.4.4 Metric for the clustering ..... 26

4.4.5 Tree construction example ..... 29

4.4.6 Properties of tree ..... 33

4.5 Singular Value Decomposition ..... 33

4.6 Rank-Revealing QR Factorization ..... 35

4.7 Projection Error ..... 37

4.8 Reduced Order Model ..... 37

4.9 Relative Error ..... 39

5 Results ........................................40

5.1 Time Function ........................................40

5.2 Bar under Tension ........................................42

5.3 Inviscid Burgers Equation ........................................53

5.3.1 Full Order Model ........................................53

5.3.2 Reduced Order Model ........................................57

5.3.3 Error computation ........................................60

6 Conclusion and Future development ........................................62

7 Appendix A. Code manual ........................................63

7.1 Tree Construction ........................................63

7.1.1 Normalization of snapshots ........................................63

7.1.2 Binary clustering ........................................64

7.1.3 Refinement Mechanism ........................................65

7.1.4 Rank revealing QR-factorization ........................................66

7.1.5 Projection error ........................................68

7.2 Reduced Order Model ........................................71

8 Appendix B. MatLab Code ........................................73

8.1 Full Order Model ........................................73

8.2 Tree construction ........................................75

8.3 Refinement mechanism ........................................79

8.4 Rank Revealing QR factorization ........................................80

8.5 Reduced Order Model ........................................81

References ........................................84

## LIST OF FIGURES

Figure 3.1. Projection error ........ 10
Figure 3.2. Inviscid Burgers equation. Shock formation ........ 15
Figure 4.1. Observations 1 and 2. Time function Not normalized (left) – highly scattered entities   Normalized (right) scatter is reduced by normalization ........ 21
Figure 4.2. Clustering of the set of points. Random scattered set of 15 points. Metric: Squared Euclidean Distance; Two clusters (left); Three clusters (right) ........ 22
Figure 4.3. Clustering of the set of points. Random scattered set of 15 points. Metric: Cosine measure; Two clusters (left); Three clusters (right) ........ 23
Figure 4.4. Clustering of the set of points. Random scattered set of 15 points. Metric: Cityblock; Two clusters (left); Three clusters (right) ........ 23
Figure 4.5. Geometric representation of clustering metrics.  Euclidean (left); Cityblock (center); Cosine (right) ........ 26
Figure 4.6. Clusters' dendrogram. Time function Not normalized (left) and normalized (right) observations ........ 30
Figure 4.7. Tree structure obtained from binary clustering. Time function. ........ 32
Figure 4.8. h-Refinement methodology for several basis ........ 35
Figure 5.1. A-priori (projection) error. Time function Not normalized (left) and normalized (right) observations. ........ 41
Figure 5.2. Geometry of the bar ........ 42
Figure 5.3. Strain-softening response curve ........ 43
Figure 5.4. Geometry of the bar with three imperfections ........ 45
Figure 5.5. Force-displacement curve of the tip. ........ 46
Figure 5.6. Singular values of the snapshot matrix. Viscoplasticity. ........ 47
Figure 5.7. A-Priori (projection) error. Viscoplasticity Not corresponding tree (left) and not normalized observations (right) ........ 48
Figure 5.8. A-Priori (projection) error. Viscoplasticity Not normalized observations (left) and RRQR not normalized observations (right) ........ 49
Figure 5.9. A-Posteriori (relative) error. Viscoplasticity Not corresponding tree (left) and not normalized observations (right) ........ 51
Figure 5.10. A-posteriori relative error for not normalized observations(left scale) and  size of splitted basis (right scale). Viscoplasticity ........ 52

Figure 5.11. Full Order Model discretized in 150 elements. Inviscid Burgers equation. ........ 54
Figure 5.12. Full Order Model discretized in 1000 elements. Inviscid Burgers equation. ...... 55
Figure 5.13. Singular values of the snapshot matrix. Inviscid Burgers equation. .................. 56
Figure 5.14. Dendrogram. Inviscid Burgers equation. ........................................................... 56
Figure 5.15. Projection Error. Inviscid Burgers equation. ...................................................... 57
Figure 5.16. ROM with 50 basis vectors. Inviscid Burgers equation. .................................... 59
Figure 5.17. ROM with 200 basis vectors. Inviscid Burgers equation. .................................. 60
Figure 5.18. Relative error of ROM. Inviscid Burgers equation. ........................................... 61

## LIST OF ALGORITHMS

Algorithm 1. Typical Newton-Raphson scheme ...................................................................... 18
Algorithm 2. Construction of snapshot matrix ........................................................................ 19
Algorithm 3. Normalization of snapshot matrix ...................................................................... 20
Algorithm 4. Typical k-means clustering ................................................................................ 25
Algorithm 5. Tree construction via binary clustering ............................................................. 27
Algorithm 6. Rank-Revealing QR Factorization ..................................................................... 36
Algorithm 7. Typical POD algorithm ...................................................................................... 37
Algorithm 8. Newton-Raphson – POD reduced ...................................................................... 38

## LIST OF TABLES

Table 5.1 . Model setup: G-FEM parameters. Strain-softening viscoplasticity. ...................... 42
Table 5.2 . Model setup: material parameters. Strain-softening viscoplasticity. ..................... 44
Table 5.3 . Model setup: boundary conditions. Strain-softening viscoplasticity. .................... 45
Table 5.4 . Model setup. Burgers Equation ............................................................................. 53
Table 5.5. ROM basis splitting into branches. Burgers Equation ........................................... 58

# 1 ABSTRACT

With a reference to a viscoplastic bar discretized by the finite element method (FEM), an h-refinement technique for model order reduction is developed. The Perzyna viscoplasticity model is adopted. The procedure employs the Proper Orthogonal Decomposition-Galerkin (POD-G) in conjunction with the h-refinement mechanism, which enriches the reduced-basis space online by ‘splitting’ a given basis vector into several vectors. For this purpose, the snapshots are collected from a standard full order FEM analysis in the offline phase and a tree structure is constructed, using a binary clustering algorithm. The singular value decomposition is used for construction of a low-dimensional basis for POD algorithm and a corresponding reduced order model (ROM). The projection error is computed offline for a predefined size of basis vector and extent of the basis splitting (refinement according to the tree). In the online phase, the truncated basis is refined based on the tree structure. The relative error is computed on the basis of full and reduced order models. The constructed reduced order model with refined basis is applied to the viscoplastic bar problem, where in the presence of imperfections (cross area reduction) giving rise to a strain-localization.

The same technique is also applied to a different problem governed by Burgers equation. Such problem, being strongly nonlinear, is an interesting benchmark for the proposed technique.

## ITALIAN ABSTRACT

Con riferimento al problema di una barra viscoplastica discretizzata utilizzando il metodo degli elementi finiti, viene sviluppata una tecnica di “h-refinement” applicata ad una procedura di “model order reduction”. Viene adottato il modello viscoplastico di Perzyna. La procedura risoltiva proposta utilizza la tecnica denominata POD-G (“Proper Orthogonal Decomposition – Galerkin”) in combinazione con il meccanismo di “h-refinement” che arricchisce lo spazio a base ridotta attraverso lo “splitting” dei vettori di base. A questo scopo, si raccolgono “snapshots” ottenuti dal modello FEM completo (generati offline) e si costruisce una struttura ad albero usando un algoritmo di “clustering” binario. Si fa uso di una decomposizione ai valori singolari per costruire una base di dimensione ridotta e il corrispondente modello di ordine ridotto. Viene valutato in modalità offline l’errore (in termini dei proiezione) conseguente alla dimensione predefinita della base adottata e del grado di “splitting” adottato. In modalità online, la base troncata viene raffinata utilizzando la struttura ad albero. L’errore relativo viene calcolato confrontando la risposta del modello completo con quella del modello a base ridotta. Il modello ridotto viene applicato al problema della barra viscoplastica in cui siano presenti delle imperfezioni (costituite da riduzioni dell’area trasversale) che generano un fenomeno di localizzazione delle deformazioni.

La stessa procedura risolutiva viene applicata anche a un problema diverso, governato dall’equazione di Burgers. Questo problema, caraterizzato da forte nonlinearità, rappresenta un banco di prova interessante per valutare l’efficacia della tecnica risolutiva proposta.

# 2 INTRODUCTION

In the modern stage of development of engineering and technology the need in large-scale numerical simulations is high. The computational cost of the high-fidelity calculation across the industries has rapidly increased in the past years. Model Order Reduction (MOR) techniques are aimed to mitigate this computational bottleneck and reduce the cost of these procedures. Moreover, for many time-critical applications computational cost of executing high-fidelity large-scale simulations remains to be infeasibly high.

The common algorithm for Reduced Order Models (ROM) consists in splitting the problem into a two-phase procedure commonly known as the offline–online decomposition. First, a computationally expensive 'offline' phase is executed. During which the training tasks, such as evaluating the high-fidelity model at several points in the input-parameter space and computation of a representative low-dimensional reduced basis for the system state are done. Second, the inexpensive 'online' phase carries out a many-query procedure. These methods use a projection process of the high-fidelity Full Order Model (FOM) equations onto the low dimensional subspace spanned by the reduced basis and quickly find the approximate solutions for arbitrary points in the input space. The implementations of these techniques are broadly applied in finite element models of solid dynamics [1], aerolasticity [2], stochastic processes [3] and fluid dynamics [4, 5].

In this research the Reduced Order Model (ROM) in conjunction with a basis refinement technique is used for a boundary value problem. A quasistatic equilibrium equation is equipped with Perzyna viscoplasticity – a nonlinear and path-dependent constitutive law [6]. The singular values of collected snapshots have a slow decay, so efficiency of the Reduced Order Model is decreased.

To improve the efficiency of ROM online, an h-refinement algorithm is developed. It consists in basis splitting according to a tree structure. The tree reflects correlation of the degrees of freedom of the system and is constructed on the basis of snapshots collected offline (in the same fashion as for typical ROM algorithm). The tree construction algorithm employed for the basis splitting is developed by means of using bottom-up binary clustering [7]. Such a clustering technique belongs to hierarchical agglomerative clustering algorithm and it links pairs of entities.

The same methodology of ROM equipped with h-refinement algorithm is applied to a problem of shock wave formation described by Burgers Equation. It is a special case of Navier-Stokes equation for Newtonian incompressible fluid. This problem is investigated in one spatial dimension and one temporal coordinate. Due to the fact that the solution of Burgers equation [8] can develop discontinuities (shock waves) it serves as a valuable benchmark for testing the ROM efficiency (and, as such, has been used as other authors to validate newly proposed MOR algorithm see [9]).

# 3 PROBLEM STATEMENT

As it is already known, in the cases of large-scale simulations the computational cost of executing is high, especially for time-critical problems. Full order models are not able to solve these problems relatively fast, so reduced order models were aimed to decrease computational burden. First, ROMs execute expensive offline stage, when the model is trained on a large dataset. Low-dimensional basis $\boldsymbol{V}^H$ is constructed, that appropriately captures the behaviour of state variable $\underline{u}$. Then, inexpensive online stage starts to compute approximate solution for arbitrary input parameters via low-dimensional subspace spanned by the reduced basis.

Being inspired by h-refinement mechanism, the proposed solution follows the similar tactics. In typical h-refinement, the domain (finite elements or volumes) is split by mesh refinement. While here, the basis vector is split according to a tree structure. Tree structure is constructed using binary bottom-up clustering. The a-priori error is computed in offline stage. The a-priori error is defined by projection of state variable lying in the low-dimensional subspace onto the full order space

$$\boldsymbol{r} = \boldsymbol{u} - \boldsymbol{u}^H \tag{3.1}$$

where $\boldsymbol{u}^H$ denotes an orthogonal projection of the vector $\boldsymbol{u}$ onto the space $V^H$ with a projection error $\boldsymbol{r}$. Graphically it is described by Figure 3.1.

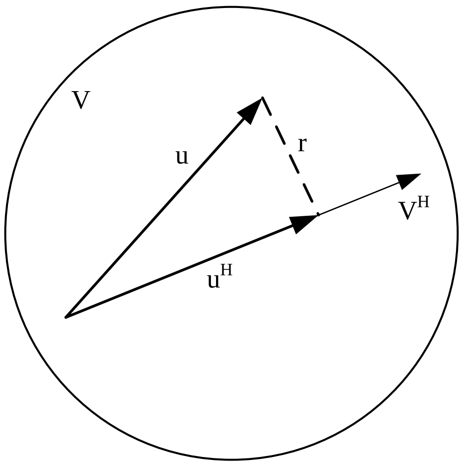


***Figure 3.1.*** *Projection error*

For a boundary value problem a parametrized system of equations of the type

$$r^k(\boldsymbol{x}^k;\boldsymbol{\mu})=0 \tag{3.2}$$

has to be solved for state of variable $\boldsymbol{x}^k \in \mathbb{R}^n$. Input parameters $\boldsymbol{\mu}$ are predefined. Residual $\boldsymbol{r}^k$ at iteration *k* has to me minimized. This is a general formulation that describes, for example, parameterized systems of linear equations arising from the finite-element discretization of elliptic PDEs. In such a case $r^k(\boldsymbol{x}^k,\boldsymbol{\mu}) \mapsto b(\boldsymbol{\mu})-a(\boldsymbol{\mu})\boldsymbol{x}$).

The output is computed as

$$\boldsymbol{z}^k = g(\boldsymbol{x}^k;\boldsymbol{\mu}) \tag{3.3}$$

With function $g:\mathbb{R}^{N_{DOF}} \times \mathbb{R}^{N_{DOF}}$ so the solution is lying in the same space as *V*, and $\boldsymbol{z}^k$ $\in \mathbb{R}^{N_{DOF}}$. When dimension of system $N_{DOF}$ is large, in full order models the outputs $\boldsymbol{z}^k$ are computed by solution of system (3.2) and then (3.3), which is computationally expensive. For time-critical problems this approach is not suitable.

## 3.1 Strain-softening Perzyna Viscoplasticity

A problem of one dimensional bar under tension equipped with viscoplastic constitutive model is employed for illustration of the ROM with h-refinement performance. A strain-softening constitutive model that is used has a following stress-strain relationship

$$\dot{\boldsymbol{\sigma}} = \boldsymbol{D}^e \, (\dot{\boldsymbol{\varepsilon}} - \dot{\boldsymbol{\varepsilon}}_{vp}) \tag{3.4}$$

with $\boldsymbol{\varepsilon}_{vp}$ is a viscoplastic strain tensor, $\dot{\boldsymbol{\varepsilon}}$ refers to time derivative of strain $\dot{\boldsymbol{\varepsilon}} = \partial\dot{\boldsymbol{\varepsilon}} / \partial t$ and $\boldsymbol{D}^e$ represents the elastic modulus tensor. The quasistatic equilibrium equation is described by

$$\boldsymbol{L}^T \boldsymbol{\sigma} = \boldsymbol{q} \; in \; \Omega \tag{3.5}$$

where a matrix of stress tensors in denoted by **σ**, $\boldsymbol{q}$ is a vector of body forces, the displacement vector $\boldsymbol{u}$ is defined in a computational domain Ω having a boundary $\Gamma = \Gamma_u \cup \Gamma_t$ with applied Dirichlet and Neumann boundary conditions. The operator matrix **L** is defined as

$$\boldsymbol{L}^T = \begin{bmatrix} \partial_x & 0 & 0 & \partial_y & 0 & \partial_z \\ 0 & \partial_y & 0 & \partial_x & \partial_z & 0 \\ 0 & 0 & \partial_z & 0 & \partial_y & \partial_x \end{bmatrix} \tag{3.6}$$

where $\partial_\# = \partial / \partial_\#$ the partial derivative with respect to #.

The flow rule of Perzyna viscoplastic model [6] is formulated as

$$\dot{\boldsymbol{\varepsilon}}_{vp} = \dot{\lambda} \frac{\partial \theta}{\partial \boldsymbol{\sigma}} \tag{3.7}$$

In this case the rate of plastic multiplier

$$\dot{\lambda} = \eta \left\langle \phi(\theta) \right\rangle^{\beta} \tag{3.8}$$

with the yield function

$$\theta = \sigma_{vm} - \sigma_Y \tag{3.9}$$

and overstress function

$$\phi(\theta) = \frac{\theta}{\sigma_Y} \tag{3.10}$$

In the above equations, ⟨#⟩ is the Macaulay bracket meaning that

$$\langle \phi(\theta) \rangle = \begin{cases} \phi(\theta) & if\ \phi(\theta) \geq 0 \\ 0 & otherwise \end{cases} \tag{3.11}$$

$\eta$ is the viscosity parameter, $\beta$ is a model parameter, $\sigma_Y$ is the current yield stress, and $\sigma_{vm}$ is the von Mises stress. To introduce a strain-softening response, relation

$$\sigma_Y = \sigma_{Y0}\ ((1 + a)\ e^{-bk} - a\ e^{-2bk}\ ) \tag{3.12}$$

is employed, where $a$ and $b$ are model parameters, $\sigma_{Y0}$ is the initial yield stress, and the plastic strain

$$k = \lambda \tag{3.13}$$

The viscoplastic strain-softening Perzyna model typically induces strongly localized strain fields.

## 3.2 INVISCID BURGERS EQUATION

In the field of solid mechanics, the differential equations in the displacement field such as

$$(\lambda + G)\ \nabla(div\ \boldsymbol{s})\ +\ G\nabla^2\boldsymbol{s}\ +\ \boldsymbol{f} = 0 \quad in \quad V \tag{3.14}$$

where $\nabla^2$ denotes Laplacian operator $(= \frac{\partial^2}{\partial^2 x} + \frac{\partial^2}{\partial^2 y} + \frac{\partial^2}{\partial^2 z}$ in Cartesian orthogonal coordinate system *xyz*) are known as the Navier equations.

The equation above is valid for a solid in a homogeneous isotropic linear elastic material. It is characterized, by the two Lamé constants, λ and G; for the homogeneity hypothesis the value of these constants is not a function of a place. For sake of simplicity, assume that the solid does not undergo inelastic deformations. Under the assumptions made above, the equations of law (in direct form) are written, at each point of the body as

$$\sigma_{ij} = \lambda I_1 \delta_{ij} + 2G\varepsilon_{ij} \tag{3.15}$$

with

$$\begin{aligned} \varepsilon_{ij} &= \frac{1}{2}(s_{i,j} + s_{j,i}) \\ I_1 &= \varepsilon_{kk} = s_{k,k} (\equiv div\, \boldsymbol{s}) \end{aligned} \tag{3.16}$$

Wishing to reach a formulation in the only unknown displacement, and replacing the indefinite equilibrium equations, it is needed to take into account the relationships between deformations and displacements

$$\left[\lambda s_{k,k}\delta_{ij} + G(s_{i,j} + s_{j,i})\right]_{,i} = -f_i \tag{3.17}$$

Assuming invertibility of the displacement components (continuous functions and regularity of the place) it becomes

$$(\lambda + G)\, s_{i,ij} + G s_{j,ii} + f_j = 0, \; j = 1,2,3 \tag{3.18}$$

in the explicit form, which is the same as compact form in the Equation (3.14). This equation is in close analogy with the indefinite equations of equilibrium of motion of viscous fluids (or Newtonian) and are reduced to so-called "Navier-Stokes" [10].

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = -\nabla\boldsymbol{p} + \mu\nabla^2\boldsymbol{u} + \boldsymbol{f} \tag{3.19}$$

were $\rho$ is the density of the fluid, $\boldsymbol{u}$ is a velocity vector field, $\boldsymbol{p}$ is the pressure, $\mu$ is a constant called the viscosity, and $\boldsymbol{f}$ is a specified external force (again a vector field).

The use of simplifications of the Navier-Stokes equations can be performed to achieve more simple versions of this equation [11], for instance dropping pressure term in case of incompressible fluids

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u}\right) = \mu\nabla^2\boldsymbol{u} + \boldsymbol{f} \tag{3.20}$$

Assuming that the external force is equal to zero and taking advantage of the fact that density $\rho$ is a constant for an incompressible fluid, this allows to define a new constant, the kinematic viscosity $\nu = \mu/\rho$ and obtain from the equation above

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = \nu\nabla^2\boldsymbol{u} \tag{3.21}$$

An even further simplification arises when we assume the viscosity is zero Then obtain the inviscid Burgers' equation:

$$\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} = 0 \tag{3.22}$$

This equation provides a useful model for many physical phenomena of a fluid flow - a nonlinear, propagating shock wave with viscous dissipation, turbulence, a propagating shock wave in gases, a propagating flame in the combustion chamber and other problems.

In the case of one-dimensional problem, the Burgers' equation (3.21) with nonzero viscosity, can be rewritten as

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 \tag{3.23}$$

or in a conservation form, when the flow is conserved:

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0 \tag{3.24}$$

where $u$ represents velocity of some quantity and the associated flow is represented by $f\,(u)$

$$f(u) = \frac{1}{2}u^2 - \nu\frac{\partial u}{\partial x} \tag{3.25}$$

The equation (3.24) is used as prototype for nonlinear hyperbolic equations and conservation laws in general [8]. It is widely used in studies of gas dynamics, turbulent fluid motion and traffic flow. Burgers equation with the characteristics

$$\frac{dx}{dt} = u \tag{3.26}$$

means that characteristic lines are straight lines in the *x, t* -plane

$$\frac{du}{dt} = 0 \tag{3.27}$$

and wave profile *u* is constant along the characteristics of the Burgers' equation.

Shock formation arises with the Burgers equation and other conservation laws and solution develops discontinuity even if the initial waveform is continuous. After a certain finite time the discontinuities may appear and then propagate in a regular manner.

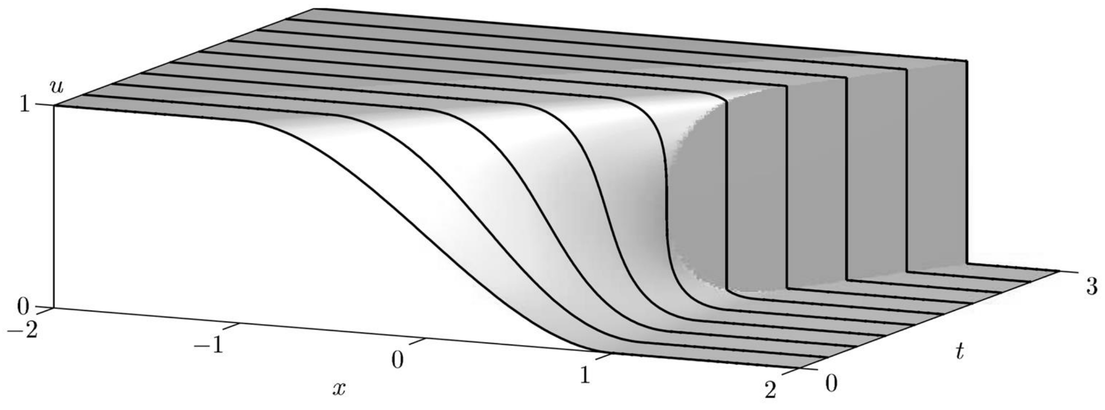


***Figure 3.2.*** *Inviscid Burgers equation. Shock formation*

It can be shown that for any kinematic viscosity $\nu > 0$, a unique smooth solution of the Equation (3.23) exists any time. The curves of Figure 3.2 are obtained by taking the limit $\nu \to 0$, and represent the case of vanishing viscosity. The Burgers equation can be considered applicable to any phenomenon of flow in which balancing effects of viscous and inertia or convective forces exist.

The solution is similar to the solution of the kinematic wave equation when inertial or convective forces predominate. It maps the boundary layers and propagating wave front. In such a case, it behaves like a hyperbolic PDE. When viscous forces predominate, the solution behaves like a parabolic equation, and any propagating wave front is blurred and dissipated due to viscous action.

Due to these different forms, which can adopt the Burgers equation in combination with its nonlinear characteristics, it has become a model for estimating and evaluating the performance of many computational methods. For this reason, the validation of ROM with h-refinement mechanism here is done employing Burgers equation for a POD-Galerkin.

Both problems of strain-softening viscoplasticity and Burgers equation require a high number of degrees of freedom for discretization of domain and solution of the system of nonlinear equations. The resolving of the gradients of strongly localized strain fields in viscoplasticity and shock formation in wave propagation implies high computational costs, since large nonlinear systems of equations of the dimensions $N_{DOF} \times N_{DOF}$ should be solved at each time step.

# 4 METHODS

## 4.1 FULL ORDER MODEL

The quasistatic equilibrium equation (3.5) can be written in a weak form and, using a Finite element formulation for space discretization, it yields to set of nonlinear equations

$$\boldsymbol{r}(\boldsymbol{a}) = \boldsymbol{f}_{\text{int}}(\boldsymbol{a}) - \boldsymbol{f}_{\text{ext}} = \boldsymbol{0} \tag{4.1}$$

where

$$\begin{aligned} \boldsymbol{f}_{\text{int}}(a) &= \int_{\Omega} \boldsymbol{B}^T \boldsymbol{\sigma}(a)\, d\Omega \\ \boldsymbol{f}_{\text{ext}} &= \int_{\Gamma_t} \boldsymbol{N}^T \overline{\boldsymbol{t}}\, d\Gamma + \int_{\Omega} \boldsymbol{N}^T \boldsymbol{q}\, d\Omega \end{aligned} \tag{4.2}$$

A matrix that contains shape functions $\boldsymbol{N} \in \mathbb{R}^{3\times N_{DOF}}$ and a matrix that contains derivatives of the shape functions $\boldsymbol{B} = \boldsymbol{L}\,\boldsymbol{N} \in \mathbb{R}^{6\times N_{DOF}}$ are substituted into (4.2). $\overline{\boldsymbol{t}}$ are the prescribed tractions on boundary $\Gamma_t$, $a \in \mathbb{R}^{N_{DOF}}$ is a vector of the nodal values of the displacement vector $\boldsymbol{u}$, and $N_{DOF}$ is the total number of degrees of freedom (DOFs).

The system of nonlinear equations is then solved using a Newton–Raphson scheme according to Algorithm 1, where at each iteration $j$ the residual vector $\boldsymbol{r}^j \in \mathbb{R}^{N_{DOF}}$ is computed using stiffness matrix $\boldsymbol{K}^j \in \mathbb{R}^{N_{DOF}\times N_{DOF}}$.

The tangential stiffness matrix is computed as

$$\boldsymbol{K}^j = \frac{\partial \boldsymbol{f}_{int}(\boldsymbol{a}_j)}{\partial \boldsymbol{a}} = \int_{\Omega} \boldsymbol{B}^T \left.\frac{\partial \boldsymbol{\sigma}}{\partial \boldsymbol{a}}\right|_{\boldsymbol{a}_j} d\Omega \tag{4.3}$$

The increment of the displacement vector $\boldsymbol{da}^{j+1} \in \mathbb{R}^{N_{DOF}}$ is computed at iteration $j + 1$.

$$\boldsymbol{D} = \left(\frac{\partial \boldsymbol{\sigma}(\Delta \boldsymbol{a})}{\partial \boldsymbol{\varepsilon}(\Delta \boldsymbol{a})}\right)^j \tag{4.4}$$

***Algorithm 1.*** *Typical Newton-Raphson scheme*

```
Input: $\Delta\boldsymbol{a}^t; \boldsymbol{D}\ ;\boldsymbol{\sigma}$

iteration $j$ = 1 , displacement at $\Delta\boldsymbol{a}^{t=1} = 0$
set $\Delta\boldsymbol{a}^1 = \Delta\boldsymbol{a}^t$, $\boldsymbol{f}_{int}^1 = \boldsymbol{f}_{int}^t$
compute new external force vector $\boldsymbol{f}_{ext}^{t+\Delta t}$
(*) compute jacobian of stress vector $\boldsymbol{D}$
compute tangent stiffness matrix at element level and
assemble tangent stiffness matrix at structure level $\boldsymbol{K}^j$
solve $\boldsymbol{K}^j \cdot d\boldsymbol{a}^{j+1} = \boldsymbol{f}_{ext}^{j+1} - \boldsymbol{f}_{int}^j(\boldsymbol{a}^j) = -\boldsymbol{r}^j$ for $d\boldsymbol{a}^{j+1}$, where $\boldsymbol{r}^j$ is a residual
update displacement vector $\Delta\boldsymbol{a}^{j+1} = \Delta\boldsymbol{a}^j + d\boldsymbol{a}^{j+1}$
compute stress $\boldsymbol{\sigma}^{j+1}$
compute internal force vector at element level
assemble internal force vector at structure level $\boldsymbol{f}_{int}^{j+1}$
$if\ \dfrac{\left\|\Delta\boldsymbol{a}^{j+1}\right\|_{L^2}}{\left\|\Delta\boldsymbol{a}^{j}\right\|_{L^2}} < tol$ then
$\Delta\boldsymbol{a}^{t+\Delta t} = \Delta\boldsymbol{a}^{j+1}$
go to next load step
else
```

```
  $j = j + 1$, go to step (*)
end if
```

```
Output: $\Delta \boldsymbol{a}^{t+\Delta t}$, $\boldsymbol{f}_{int}^{t+\Delta t}$
```

## 4.2 DATA COLLECTION

The results of the solution of Full Order Model (FOM) or generally speaking the observation data is used as a training set for offline stage. The matrix of snapshots $\boldsymbol{X} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$ collects data in a format where rows contain degrees of freedom and columns the snapshot number, and it has a dimensionality $N_{Samp} \times N_{DOF}$. Typically, the snapshot collection is organized according to the Algorithm 2.

***Algorithm 2.*** *Construction of snapshot matrix*

```
Input: $N_{Samp}$ – number of desired snapshots at time $T$

set $\boldsymbol{X}$ = [ ]
repeat
    run full order model simulation for $t$ time steps
    $\boldsymbol{X} \leftarrow [\boldsymbol{X}\ \boldsymbol{X}^t]$
    $t = t + \Delta t$
until $t = T$

Output: snapshot matrix $\boldsymbol{X} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$
```

## 4.3 Data Normalization

For the scope of capturing the correlation of the degrees of freedom, it is possible to normalize training set according to Equation (4.5) inherited into Algorithm 3. In this case the correlated and anticorrelated state variables become separated by small Euclidean distance and can be grouped via a clustering technique.

$$\begin{aligned} x_{i,j} &\leftarrow \frac{x_{i,j}}{\left\| \boldsymbol{x}_i^T \right\|_{L^2}} \\ \boldsymbol{x}_i^T &\leftarrow -\boldsymbol{x}_i^T;\ if\ x_{i1}^T < 0 \end{aligned} \tag{4.5}$$

where $\|.\|_{L^2}$ denotes a Euclidean ($L^2$) norm calculated by

$$\|\boldsymbol{X}\|_{L^2} = \sqrt{\sum_{i=1}^{N} \sigma_i^2} \tag{4.6}$$

where $\sigma_i$ denotes the singular values of $\boldsymbol{X}$ and $N$ is a length of vector $\boldsymbol{x}_i$.

***Algorithm 3.*** *Normalization of snapshot matrix*

```
Input: Snapshot matrix $\boldsymbol{X} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$, containing $N_{Samp}$ observations

for i=1,...,$N_{DOF}$

        Normalize rows of $X$ to capture correlation and anti-correlation by clustering

                $x_{i,j} \leftarrow x_{i,j} / \left\| \boldsymbol{x}_i^T \right\|$

        Flip origin if the first entity is negative

                $\boldsymbol{x}_i^T \leftarrow -\boldsymbol{x}_i^T;\ if\ x_{i1}^T < 0$

end

Output: $\boldsymbol{X} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$, normalized
```

The difference in scatter is visible in Figure 4.1 when, for instance taking the input from snapshot matrix of time function (4.10), plotting first taken snapshot versus the second one. In Figure 4.1 (left) the scatter is high and the anti-correlation is not seen, while the Figure 4.1 (right) shows the same dataset after normalization. The 3 clusters are clearly seen.

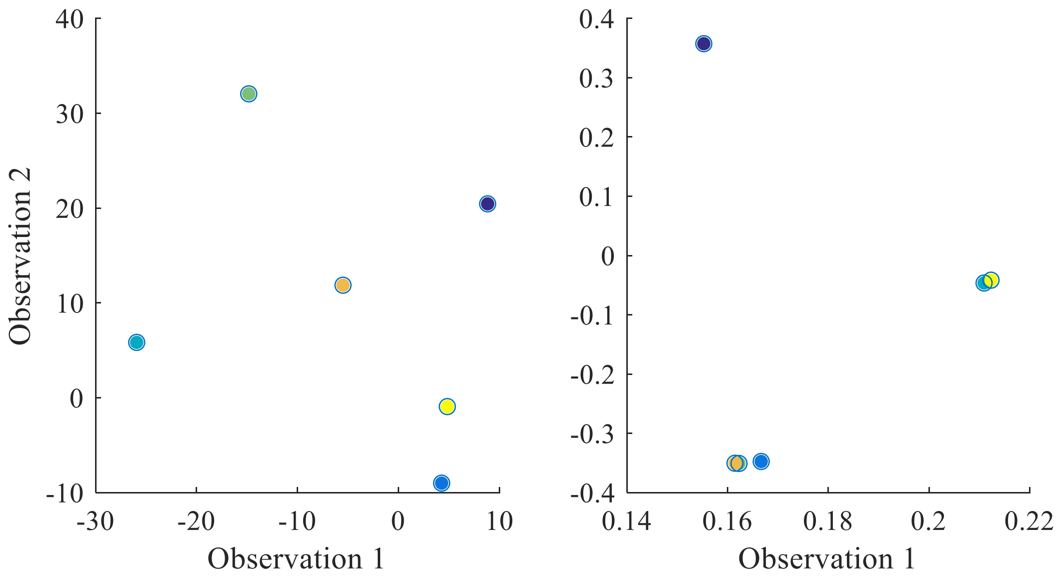


***Figure 4.1.*** *Observations 1 and 2. Time function*

*Not normalized (left) – highly scattered entities*

*Normalized (right) scatter is reduced by normalization*

## 4.4 TREE CONSTRUCTION

### 4.4.1 *CLUSTERING*

The data related to the scatter of snapshot matrix and capturing of the clusters information is used for a tree construction. For this purpose, a clustering algorithm has to be employed. There exist several possibilities to do this. The first one consists in algorithm when the entire set of observations is split into the clusters, known as k-means clustering. The existing challenge of this method is the definition of the number of clusters required. The proposed here solution can be seen with two conceptually different approaches: top-down and bottom-up. The first case is consisted in splitting the entire set in 2 clusters recursively, while the second links pair of entities combining them into the binary clusters.

### 4.4.2 CLUSTERING TYPOLOGY

There exist two conceptually different types of clustering techniques: hierarchical and partitional. Partitional clustering is a division of the set into the non-overlapping subsets, such that the entity belongs to exactly one subset.

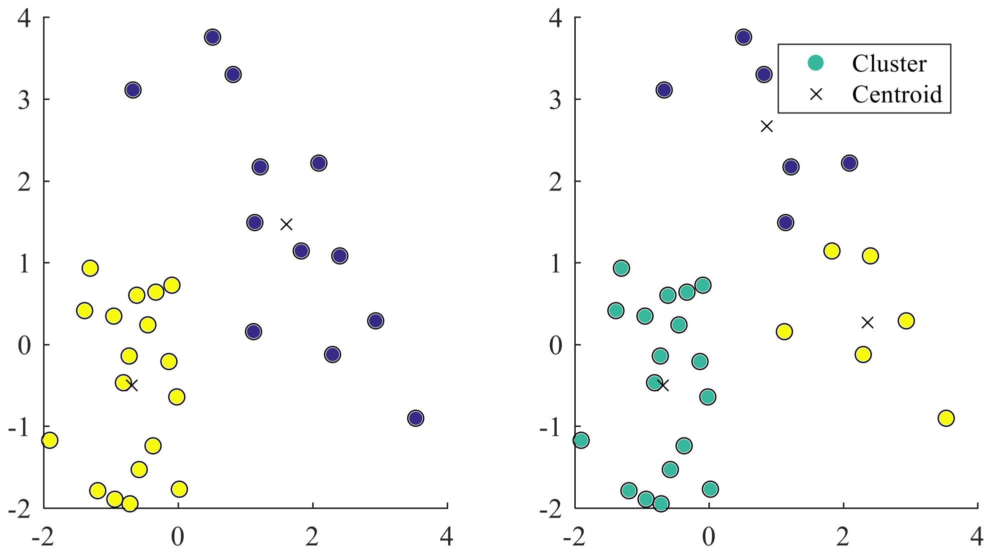


***Figure 4.2.*** *Clustering of the set of points. Random scattered set of 15 points. Metric: Squared Euclidean Distance; Two clusters (left); Three clusters (right)*

Taking individually, each association of the points into a group from Figure 4.2 to Figure 4.4 is a partitional clustering. If the clusters contain sub-clusters, then we obtain hierarchical clustering, when the clusters are nested and organized as a tree. Each node of the tree except the bottom level (leaf nodes) is association of child sub-clusters. The root of the tree (top cluster) contains all the entities, the set consists of. The example of illustration of such a tree organized in a form of dendrogram would be demonstrated in Chapter 4.4.5.For example, on the Figure 4.4 (left), the blue cluster consists in union of blue and yellow Figure 4.4 (right), allowing the further splitting. The yellow cluster on the Figure 4.4 (left) is completely coincident to the green of Figure 4.4 (right). The hierarchical clustering can be viewed as a sequence of partitional clustering and vice versa, partitional clustering can be obtained by taking any member of the hierarchical sequence, cutting the tree at a particular level.

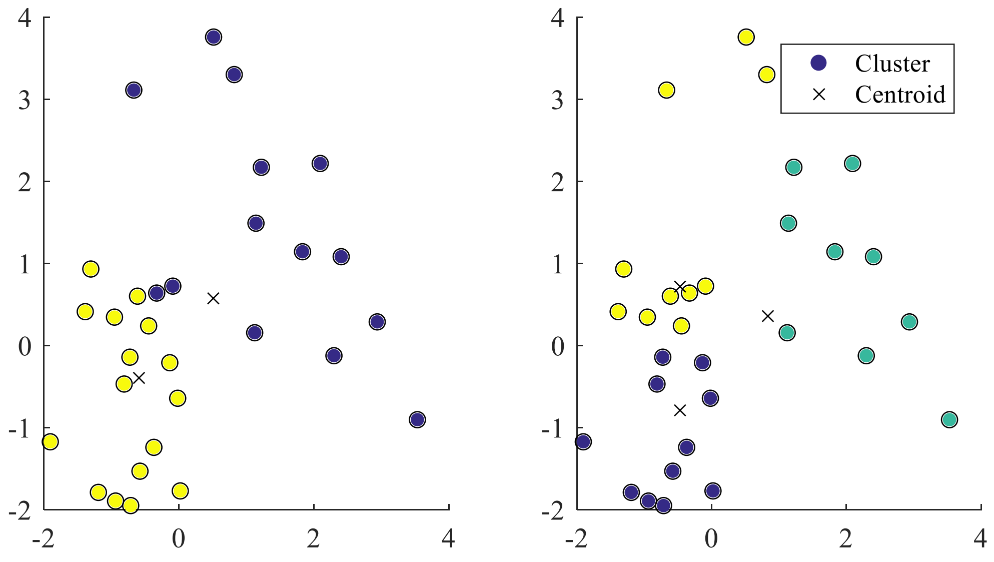


***Figure 4.3.*** *Clustering of the set of points. Random scattered set of 15 points. Metric: Cosine measure; Two clusters (left); Three clusters (right)*

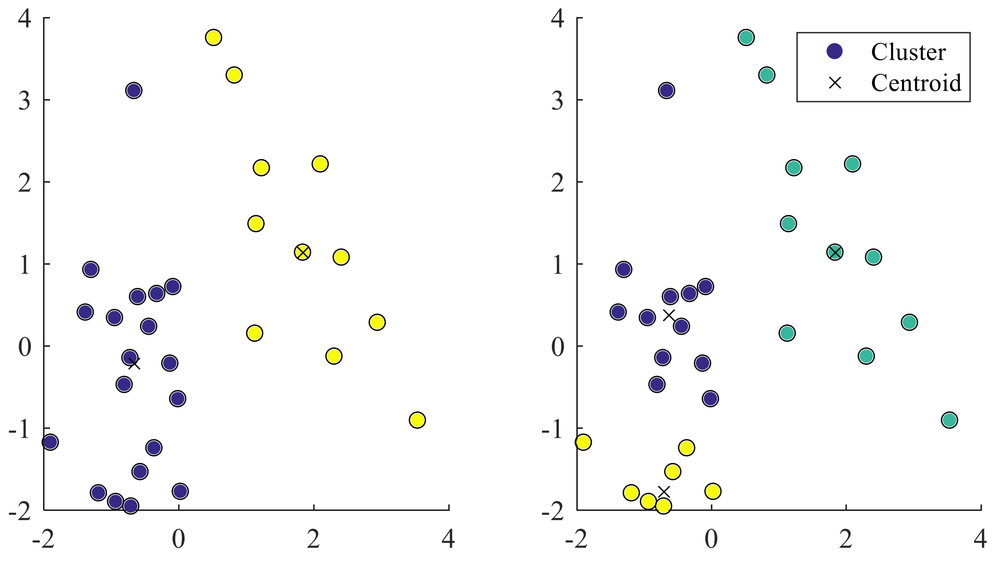


***Figure 4.4.*** *Clustering of the set of points. Random scattered set of 15 points. Metric: Cityblock; Two clusters (left); Three clusters (right)*

Complete clustering assigns each entity to a cluster. Partial clustering may not include some elements to a set. This may happen when data in some sets does not belong to any well-defined group. In many cases this data represents noise outliers or background objects, not contributing to a scope.

Another way to distinguish the clustering is an exclusive, when an entity belongs to a single cluster, or overlapping when a point can be placed in several clusters. Non-exclusive (overlapping) reflects the fact that an object can simultaneously belong to more than one class. In fuzzy clustering a weight factor is assigned to each variable and it varies from 0 (absolutely does not belong) to 1 (fully belongs). In a same fashion, probabilistic clustering computes probability with which the entity belongs to a cluster, and probabilities must also sum to a unit.

Practically, the probabilistic and fuzzy clustering to can be converted into exclusive clustering, when the element is assigned to a cluster with the highest probability or weight.

Prototype-Based Algorithms are aimed to learn a prototype for each cluster, and form clusters by data objects around the prototypes [7]. For vast majority of the algorithms of this class, the prototype is a centroid of a cluster, and the clusters tend to be globular.

Graph-Based Algorithms regard data objects as nodes, and the distance between two objects as the weight of the edge connecting the two nodes, the data can be represented as a graph, and a cluster can be defined as a connected subgraph. In a typical graph-based algorithm defines the shared nearest-neighbours for each data object, and then sparsifies the graph to obtain the clusters.

Density-Based Algorithms take a cluster as a dense region of data objects that is surrounded by regions of low densities. They are often employed when the clusters are irregular or intertwined, or when noise and outliers are present. In case of highly dimensional data, the density notion is valid only in subspaces of features, which motivates the subspace clustering.

### 4.4.3 *K-MEANS CLUSTERING*

The one of the most widely used and oldest clustering algorithms is *k*-means clustering. It is a prototype-based partitional clustering algorithm [7] that attempts to find *k* non-overlapping clusters. These clusters are represented by their centroids (a cluster centroid is typically the mean of the points in that cluster).

***Algorithm 4.*** *Typical k-means clustering*

```
Input: dataset X, number of clusters k
Select k initial centroids
    repeat
        form k clusters by assigning each point to the closest centroid
        recalculate centroid of each cluster
    until position of centroids do not change
Output: k cluster sets and their centroids with the points belonging to the cluster
```

The clustering process of *k*-means is described by Algorithm 4. First, *k* initial centroids are selected, where *k* is specified in the input and indicates the desired number of clusters. Every point in the data is then assigned to the closest centroid, and each collection of points assigned to a centroid forms a cluster. The centroid of each cluster is then updated based on the points assigned to that cluster. This process is repeated until no point changes clusters.

The examples of application of *k*-means clustering for a random dataset using different metrics from the Chapter 4.4.4 are demonstrated in Figure 4.2, Figure 4.3 and Figure 4.4

### 4.4.4 *Metric for the clustering*

In order to link pairs of vectors it is needed to establish a metric for the pairwise distance [12]. The way of capturing similarities between observations in the examples is the Euclidean distance

$$d^2_{ij} = (x_i - x_j)(x_j - x_j)' \tag{4.7}$$

where for a given matrix of snapshots $X \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$ , the $N_{Samp}$ row vectors have the various distances between $x_i$ and $x_j$. There exist another ways of computing the distance, as for instance cosine similarity evaluated for vectors $x_i$ and $x_j$ as

$$d_{ij}(x_i, x_j) = 1 - \cos(x_i, x_j) = 1 - \frac{x_i x_j'}{\sqrt{(x_i x_i')(x_j x_j')}} \tag{4.8}$$

The metric such as Cityblock

$$d(x,c) = \sum_{j=1}^{p} \left| x_j - c_j \right| \tag{4.9}$$

is defined as sum of absolute differences ($L^1$ distance). Each centroid $c_j$ is the component-wise median of the points $x_j$ in that cluster.

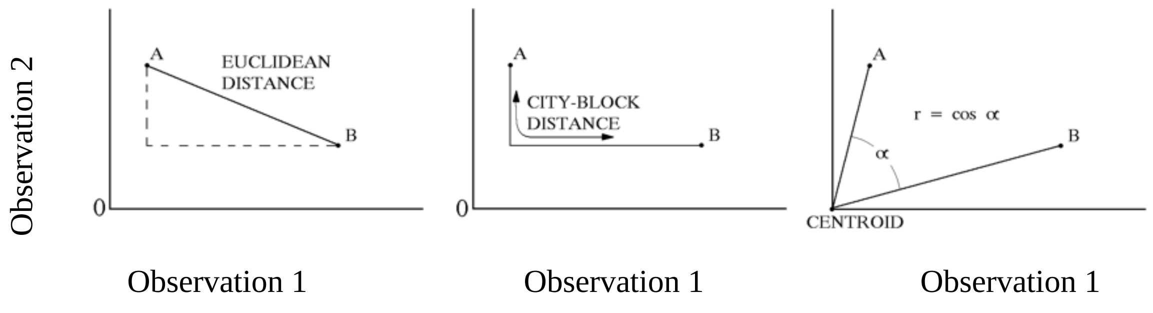


***Figure 4.5.*** *Geometric representation of clustering metrics. Euclidean (left); Cityblock (center); Cosine (right)*

Geometric representations of basic distance measures between two entities (A and B) in 2D space are shown in Figure 4.5. Looking at 2D plain of Cartesian coordinate system, the Euclidean distance is computed is a length of AB, the axes in Figure 4.5 (left) meet at the origin. In the Figure 4.5 (center) Cityblock distance is computed as the sum of projections of AB onto the orthogonal coordinate system in, that is why it also sometimes known as Manhattan distance. In most cases, this distance measure yields to results similar to the Euclidean distance. However, Cityblock distance provides the effect of a large difference when a single dimension is dampened, since the distances are not squared. Cosine distance in Figure 4.5 (right) is measured by (4.8), where similarity r = cos(α) is computed for vectors taking the origin from centroid.

It is observed in [13] that having large dimensionality of vectors, the difference between results of computation Euclidean (4.7) or cosine (4.8) distance is not significant. On the other hand, the studies such as [14] point out that squared Euclidean distance is not suitable for high-dimensional *k*-means data clustering because of the "curse of dimensionality". Nevertheless, investigation of this problem for different datasets is interesting task to deal with in the future research.

***Algorithm 5.*** *Tree construction via binary clustering*

```
Input: Snapshot matrix $\boldsymbol{X} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$, can be normalized

compute Pairwise (Euclidean) Distance from snapshot matrix between vectors $x_i$ and $x_j$
        $d^2_{ij} = (x_i - x_j)(x_i - x_j)'$
grouping into binary clusters by creation agglomerative hierarchical cluster tree
        compute the proximity matrix *LinkX*, based on a metric
        *repeat*
                merge the closest two clusters
                update the proximity matrix *LinkX* to reflect the metric $d_{ij}$ between the
                centroids of new cluster and the original clusters
```

```
    *until*      only one cluster remains
create first $N_{DOF}$ clusters corresponding to observations themselves
    *for i=1,…,$N_{DOF}$*
        *Tree*{*1,…,$N_{DOF}$*}*=1,…,$N_{DOF}$*
    *end*
create definition of each cluster *EEE*
    *for i=1,…,size(LinkX)*
        *take* left cluster of *LinkX*, define its content by entities
            ResL=E{LinkX (i,1)}
        *take* right cluster of *LinkX*, define its content by entities
            ResR=E{LinkX(i,2)}
        *combine* defined left and right clusters of *LinkX*
            E{i}=[ ResL ResR]
    *end*
*for i=1,…,size(Tree)*
    *replace* cluster name by its definition (contents)
        TREE{i}= LinkX{ E{i}}
*end*
```

Output: *TREE* – String of arrays $N_{DOF} \times 1$ containing trees for each branch (For example see Figure 4.7)

### 4.4.5 *Tree construction example*

It is necessary to clarify Algorithm 5 with a comprehensive example. “Time function” is the example of data collected from time history analysis [15]. Snapshot matrix

$$X = \begin{matrix} 8{,}8078 & 20{,}3701 & -10{,}6956 & -37{,}2035 & 25{,}7282 & 12{,}9818 & -16{,}8462 & 12{,}9854 \\ 4{,}2437 & -9{,}1066 & 6{,}2987 & -9{,}6227 & 13{,}0436 & -16{,}0915 & 3{,}5370 & 0{,}1259 \\ -25{,}8879 & 5{,}6943 & 26{,}4645 & 47{,}9825 & 24{,}0329 & 20{,}2440 & 83{,}5343 & -57{,}7367 \\ -14{,}8654 & 31{,}9001 & -22{,}0641 & 33{,}7077 & -45{,}6912 & 56{,}3677 & -12{,}3898 & -0{,}4409 \\ -5{,}5002 & 11{,}8031 & -8{,}1638 & 12{,}4720 & -16{,}9059 & 20{,}8562 & -4{,}5842 & -0{,}1631 \\ 4{,}8868 & -1{,}0749 & -4{,}9956 & -9{,}0575 & -4{,}5366 & -3{,}8214 & -15{,}7684 & 10{,}8987 \end{matrix} \tag{4.10}$$

contains $N_{Samp}$ = 8 snapshots and $N_{DOF}$ = 6 degrees of freedom.

The snapshot matrix ***X*** (4.10) of a time function after processing by means of normalization becomes more suitable for clustering (see Figure 4.1 (right)). Binary clustering is adopted to capture correlations between degrees of freedom. The Euclidean distance is computed at the each iteration, defining the metric for linking pairs of clusters. It creates the matrix ($2N_{DOF}$ -1) × 3

| Global Cluster | Left Cluster | Right Cluster | Distance |
|---|---|---|---|
| {1} | 1 | 1 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| {6} | 6 | 6 | 0 |
| {7} | 4 | 5 | 0.0039 |
| {8} | 2 | 7 | 0.0062 |
| {9} | 3 | 6 | 0.0071 |
| {10} | 1 | 9 | 1.0083 |
| {11} | 8 | 10 | 1.3088 |

(4.11)

where the Global Cluster contains the Left and Right clusters, telling the linked pair. The last column defines the value of the metric which was used for creating a new cluster. At the next stage the names of the clusters are replaced with the clusters form *1* to $N_{DOF}$ themselves, for instance Global Cluster {10}, containing Global Clusters {1} as Left Cluster and {9} as a Right Cluster becomes {1, 3, 6} after substitution.

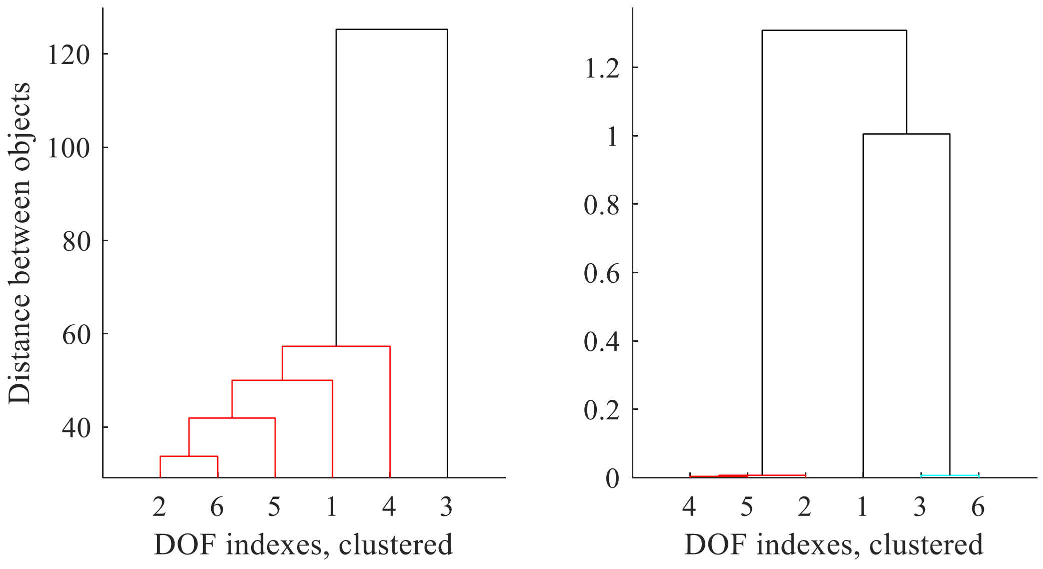


***Figure 4.6.*** *Clusters' dendrogram. Time function*
*Not normalized (left) and normalized (right) observations*

As it was discussed before, traditional k-means clustering can be used instead of bottom-up algorithm. For both algorithms the optimal solution is depicted at Figure 4.6, which is a dendrogram, describing the tree structure of observations. Dendrogram is a tree structure, where each node is associated with a distance (height) and satisfies

$$h(A) \le h(B) \Leftrightarrow A \subseteq B \tag{4.12}$$

where *h*(*A*) and *h*(*B*) denote the heights of *A* and *B* respectively. For all subsets of points *A* and *B* if $A \cap B \neq \varnothing$ is necessary and sufficient

$$h_{ij} \le \max\{h_{ik}, h_{jk}\}, \forall i, j, k \in [n] \tag{4.13}$$

where $h_{i,j}$ denotes the height of internal node specifying the smallest cluster to which both $x_i$ and $x_j$ belong, for each pair of data points ( $x_i, x_j$ ).

Figure 4.6 (left) figure describes the dendrogram associated with the Figure 4.1 (left), when the normalization is not done. The right dendrogram is constructed on the basis of normalized snapshots. Conceptually, the refinement tree branches can be determined at each desired distance between the observations. The needed depth level of tree could be graphically obtained by intersection of horizontal line at the predefined distance and Degree of Freedom index.

Each association of the adjacent state variables create a new cluster, e.g. branch. This branch is united with the next closest observation or branch, which has the closest centroid. The initial clusters correspond to number of degrees of freedom, i.e. correspond to itself, so the distance is zero. The graphical representation of creation of the new clusters corresponds to Figure 4.7, where the new created clusters are listed at (4.11). The complete procedure of the new clusters creation consists in assigning to the $N_{Samp}$ observations, the clusters containing themselves and then estimating the Euclidean distance between the clusters (observations). The succeeding procedure is based on the creation of the binary clusters by collecting them on the basis of distance between centroids.

The refinement mechanism is based on the tree structure in Figure 4.7, where the needed depth of refinement can be obtained by the choice of the needed level of the tree. The array of the tree structure (4.14) contains matrix of the elements required for splitting on the each level of the basis refinement such as

| String Number | Array Contents |
|---|---|
| 1 | $\{1,...,6\}$ |
| 2 | $\{1,3,6\}\{2,4,5\}$ |
| 3 | $\{1\}\{3,6\}\{2,4,5\}$ |
| 4 | $\{1\}\{3\}\{6\}\{2,4,5\}$ |
| 5 | $\{1\}\{3\}\{2,4\}\{5\}\{6\}$ |
| 6 | $\{1\}\{2\}\{3\}\{4\}\{5\}\{6\}$ |

(4.14)

where the string of arrays contains the basis splitting for the chosen branch of the tree.

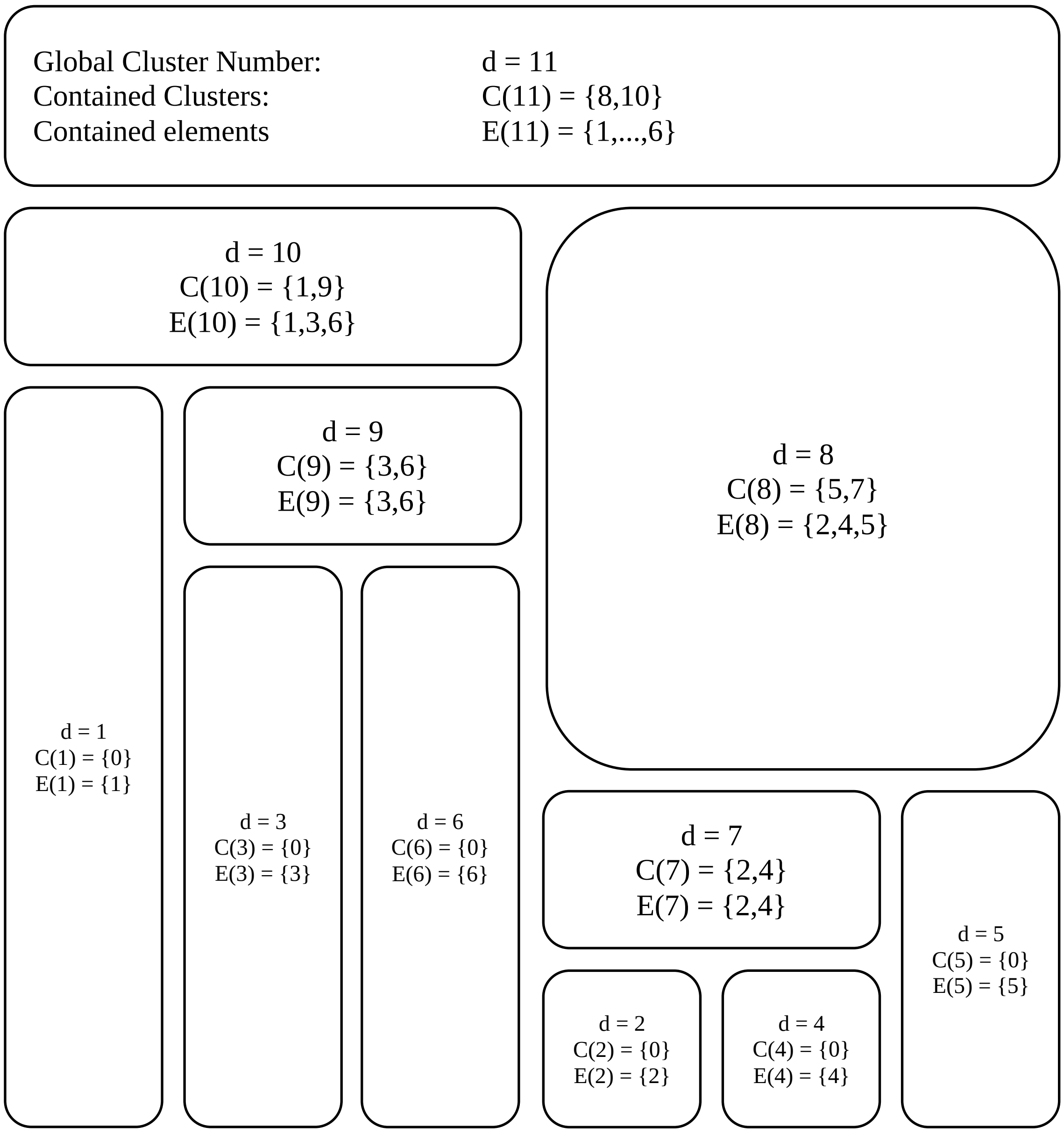


***Figure 4.7.*** *Tree structure obtained from binary clustering. Time function.*

On one hand, for a large dataset, the binary bottom-up clustering can be more computationally demanding than the *k*-means clustering. Due to the fact that during the offline stage the computational cost is not critical, the more expensive binary clustering is acceptable for a tree construction. On the other hand, the choice of the number of clusters for k-means clustering is questionable. Moreover, the iterative procedure of *k*-means clustering such as Algorithm 4 does not always converge to the optimal minimum.

### 4.4.6 *PROPERTIES OF TREE*

1. Hierarchical subspaces

Refinement method with the use of a tree creates a hierarchy of subspaces such as $range(\boldsymbol{V}^H) \subseteq range(\boldsymbol{V}^h)$.

The poof of this property is:

$$range(\boldsymbol{V}^H) = \left\{\boldsymbol{V}^h w \mid w \in range(I_H^h) \subseteq \mathbb{R}^q\right\} \subseteq \left\{\boldsymbol{V}^h w \mid w \in range(\boldsymbol{I}_H^h) \subseteq \mathbb{R}\boldsymbol{q}\right\} = range(\boldsymbol{V}^h)$$

2. Fully refined ROM basis converges to the FOM solution

If every element has non-zero entity in one of the original reduced-basis vectors,

$\forall l \in \mathbb{N}(n),\ \exists (i,j) \in \mathbb{N}(n) \times \mathbb{N}(p^{(0)}) \mid v_{ij}^{(0)} \neq 0$ and

$\forall l \in \mathbb{N}(n),\ \exists i \in \mathbb{N}(m) \mid E(i) = l, C(i) = \varnothing$ holds, this means that fully refined basis converges to Full Order Model solution.

Both properties 1 and 2 [15] say that with refinement a sequence of hierarchical subspaces can be generated. The ROM converges to full-order model solution when the basis is completely split. However, the use of this methodology could produce rank-deficient matrices, when a completely split basis has linearly dependant columns. In order to be able to detect and remove rank-deficiency, the refinement method proposes the use of rank-revealing QR factorisation (see Chapter 4.6) after each split of basis.

## 4.5 SINGULAR VALUE DECOMPOSITION

The matrix of the snapshots $\boldsymbol{X}$ which has dimensions of $N_{Samp} \times N_{DOF}$ can be seen as

$$\underset{N_{DOF} \times N_{Samp}}{\boldsymbol{X}} = \underset{N_{DOF} \times N_{DOF}}{\boldsymbol{V}} \ \underset{N_{DOF} \times N_{Samp}}{\boldsymbol{\Sigma}} \ \underset{N_{Samp} \times N_{Samp}}{\boldsymbol{W}^T} \tag{4.15}$$

The operation described by the Equation (4.15) is called Singular Value Decomposition (SVD) with the following components:

$\underset{N_{DOF}\times N_{DOF}}{\boldsymbol{V}}$ - Left singular matrix defined by the eigenvectors of $\boldsymbol{X}\ \ \boldsymbol{X}^T$; orthonormal

$\underset{N_{Samp}\times N_{Samp}}{\boldsymbol{W}^T}$ - Right singular matrix defined by the eigenvectors of $\boldsymbol{X}^T\ \boldsymbol{X}$ ; orthonormal

$\underset{N_{DOF}\times N_{Samp}}{\boldsymbol{\Sigma}}$ - Singular basis matrix, containing the $\sigma_{11} > \sigma_{22} > .. > \sigma_{NN}$ ; diagonal

The diagonal values of the matrix Σ contain the variance of the taken snapshots. The number of significant values of σ is showing the number of principal components $N_{P.C}$ of the snapshot matrix $X$

The three components ***V***, **Σ**, $\boldsymbol{W}^T$ of the singular value decomposition are truncated according to $N_{P.C}$

For the certain value of $\boldsymbol{f}_{ext}$, the solution in terms of displacements ***a(x)*** can be found by the same algorithm, such as Newton-Raphson, but solving the reduced system.

The further application of SVD is discussed on the basis of h-refinement for a tree construction in a Chapter 4.4. Taking just the first column from the left-side singular vector

$$\boldsymbol{V}^{H^T} = 0.2606 \quad 0.0355 \quad -0.9391 \quad -0.1242 \quad -0.0460 \quad 0.1773 \tag{4.16}$$

obtained from the singular value decomposition of snapshot matrix (4.10), and constructing tree matrix according to the tree structure from the Figure 4.7 for *branch* = 3, it becomes

$$\boldsymbol{E}\{3\} = \begin{matrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 4 & 5 & 0 \\ 0 & 0 & 3 & 0 & 0 & 6 \end{matrix} \tag{4.17}$$

the matrix containing the location of cluster contents in the refined basis (see Algorithm 5). The obtained reduced basis with the arrangement according to the tree structure is

$$\boldsymbol{V}^{h^T} = \begin{matrix} 0.2606 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.0355 & 0 & -0.1242 & -0.0460 & 0 \\ 0 & 0 & -0.9391 & 0 & 0 & 0.1773 \end{matrix} \quad (4.18)$$

Here $\boldsymbol{V}^H$ and $\boldsymbol{V}^h$ denote refined and unrefined basis accordingly. In the same manner it is possible to refine more basis vectors as it is illustrated by Figure 4.8.

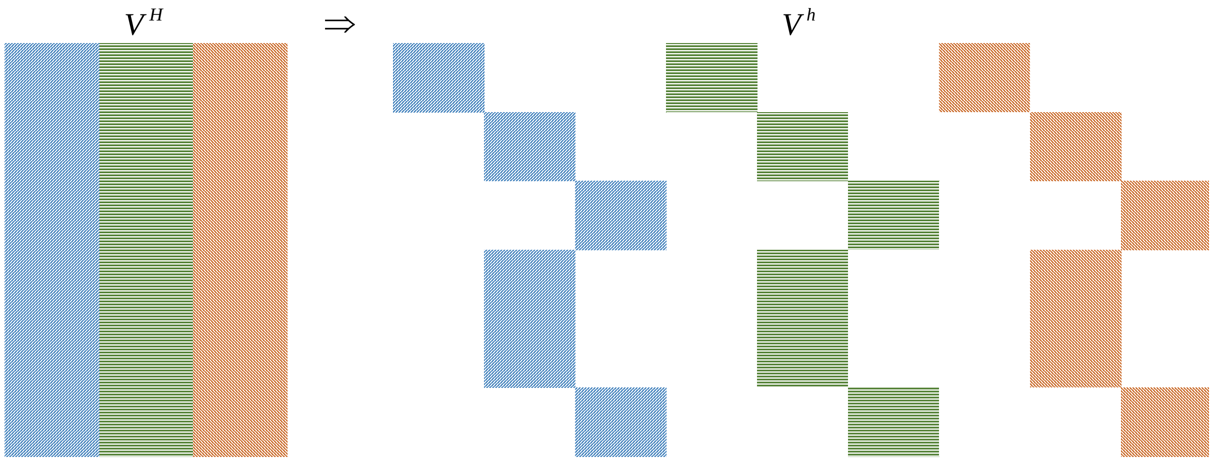


***Figure 4.8.*** *h-Refinement methodology for several basis*

## 4.6 RANK-REVEALING QR FACTORIZATION

Rank revealing QR factorization consists in matrix decomposition algorithm based on QR factorization. The aim is to determine rank deficiency of matrix [16, 17]

$$\boldsymbol{X} = \boldsymbol{QR} = \boldsymbol{Q}\begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix} = \boldsymbol{Q}\begin{bmatrix} \boldsymbol{R}_1 \\ 0 \end{bmatrix} = \boldsymbol{Q}_1\boldsymbol{R}_1 \quad (4.19)$$

where $\boldsymbol{R}_1$ is an $n\times n$ upper triangular matrix, $0$ is an $(m-n)\times n$ zero matrix, $\boldsymbol{Q}_1$ is $m\times n$, $\boldsymbol{Q}_2$ is $m\times(m-n)$, and $\boldsymbol{Q}_1$ and $\boldsymbol{Q}_2$ both have orthogonal columns. The typical iterative procedure is defined by the Algorithm 6

***Algorithm 6.*** *Rank-Revealing QR Factorization*

```
Input: Rank-deficient refined basis $\boldsymbol{V}^h \in \mathbb{R}^{N \cdot branch \times N_{DOF}}$, Tolerance $\varepsilon$

Compute $L^2$ norm of first column
        $\rho = \left\| \boldsymbol{V}^h(:,1) \right\|_{L^2}$
Store first normalized column
        $\boldsymbol{Q} = \boldsymbol{q}_1 = \frac{\boldsymbol{V}^h(:,1)}{\rho}$
for $i$= 2 : size($\boldsymbol{V}^h$, 2)
    Compute
        $\boldsymbol{r} = \boldsymbol{Q}^T \cdot \boldsymbol{V}^h(:, i)$
        $\boldsymbol{q}_i = \boldsymbol{V}^h(:, \mathrm{i}) - \boldsymbol{Q} \cdot \boldsymbol{r}$
        $\rho_i = \left\| \boldsymbol{q}_i \right\|_{L^2}$
        if $\frac{\rho_i}{\left\| \boldsymbol{r}_i \right\|_{L^2}} < \varepsilon$
            store index of linearly dependant column
                $j_{out} = [\, j_{out}, i \,]$
        else
            proceed appending the columns to matrix $\boldsymbol{Q}$
                $\boldsymbol{Q} = [\boldsymbol{Q}, \boldsymbol{q}_i / \rho\,]$
        end if
end

Output: $j_{out}$ – vector containing indexes of linearly dependant columns
```

## 4.7 PROJECTION ERROR

After removal of linear dependency of the columns of left orthogonal vector the basis vectors are processed through RRQR, the projection error can be computed as

$$\frac{\left\| \boldsymbol{X} - \boldsymbol{V}^h (\boldsymbol{V}^h)^{\dagger} \boldsymbol{X} \right\|_F}{\left\| \boldsymbol{X} \right\|_F} \tag{4.20}$$

where † denotes Moore–Penrose pseudoinverse , $\left\| \cdot \right\|_F$ is a Frobenious norm, $\boldsymbol{X}$ corresponds to the initial dataset, on the basis of which the tree was constructed and $\boldsymbol{V}^h$ is a refined basis vector.

## 4.8 REDUCED ORDER MODEL

For typical ROM the quality of training set plays a crucial role in the accuracy of solution. Usually the span of taken snapshots has to contain the information about possible events, since the solution of something that was not observed during the training cannot be obtained accurately.

The matrix of observations $\boldsymbol{X}$ is used for construction of the left orthogonal vector by means of SVD (see Chapter 4.5) and used for instance in the POD basis construction such as Algorithm 7

***Algorithm 7.*** *Typical POD algorithm*

```
Input: Snapshot matrix $\boldsymbol{X} \in \mathbb{R}^{N_{samp} \times N_{DOF}}$

Compute SVD: $\boldsymbol{X} \in \boldsymbol{V} \cdot \boldsymbol{\Sigma} \cdot \boldsymbol{W}^T$
Choose number of basis vectors $n \in \{1, 2, ...n\}$
Truncate basis $\boldsymbol{V}^H = [\boldsymbol{V}^1 ... \boldsymbol{V}^n]$ $where \boldsymbol{V} = [\boldsymbol{V}^1 ... \boldsymbol{V}^{N_{DOF}}]$

Output: $\boldsymbol{V}^H \in \mathbb{R}^{n \times N_{DOF}}$
```

The iterative finite element model procedure of reduced order model described by Algorithm 8 similar to the full order model discussed in Chapter 4.1.

***Algorithm 8.*** *Newton-Raphson – POD reduced*

```
Input: $\Delta\boldsymbol{a}_r^t; \boldsymbol{D}_i; \boldsymbol{\sigma}$

Iteration $j = 1$, displacement at $\Delta a_r^{t=1} = 0$
Set $\Delta\boldsymbol{a}_r^1 = \Delta\boldsymbol{a}_r^t$, $\boldsymbol{f}_{int,r}^{1} = \boldsymbol{f}_{int,r}^{t}$
compute new external force vector $\boldsymbol{f}_{ext,r}^{1} = \boldsymbol{f}_{ext,r}^{t+\Delta t}$
(*) compute jacobian of stress vector $\boldsymbol{D}_r$
compute tangent stiffness matrix at element level
assemble tangent stiffness matrix at structure level $\boldsymbol{K}^j$
compute $\boldsymbol{K}_r^j$
solve $\boldsymbol{K}_r^j \cdot d\boldsymbol{a}_r^{j+1} = \boldsymbol{f}_{ext,r} - \boldsymbol{f}_{int,r}^{j}$ for $d\boldsymbol{a}_r^{j+1}$
update displacement vector $\Delta\boldsymbol{a}_r^{j+1} = \Delta\boldsymbol{a}_r^j + d\boldsymbol{a}_r^{j+1}$
compute stress $\boldsymbol{\sigma}_r^{\ j+1}$
compute internal force vector at element level
assemble internal force vector at structure level $\boldsymbol{f}_{int,r}^{j+1}$
project $\Delta\boldsymbol{a}_r$ , $\boldsymbol{f}_{ext,r}$ and $\boldsymbol{f}_{int,r}$ back to full order space
if $\left\|\Delta\boldsymbol{a}^{j+1}\right\|_{L^2} < tol \cdot \left\|\Delta\boldsymbol{a}^{j}\right\|_{L^2}$
then $\Delta\boldsymbol{a}^{t+\Delta t} = \Delta\boldsymbol{a}^{j+1}$
go to next load step
```

```
else
    $j = j + 1$ , go to step (*)
end if
Output: $\Delta \boldsymbol{a}_r^{t+\Delta t}$, $\boldsymbol{f}_{int,r}^{\,t+\Delta t}$
```

## 4.9 Relative Error

For the chosen refined basis $\boldsymbol{V}^h$ the online stage is executed through POD algorithm (see Algorithm 8) and relative error is computed as

$$a - posteriori\, error = \frac{1}{t}\sum_{k=1}^{t} \frac{\left\| u_{FOM}(\cdot,\tau^k) - u_{ROM}(\cdot,\tau^k) \right\|_{L^2}}{\left\| u_{FOM}(\cdot,\tau^k) \right\|_{L^2}} \tag{4.21}$$

where of difference between output of ROM $u_{ROM}$ and FOM $u_{FOM}$ is normalized using $L^2$ norm at the each time step $\tau$

# 5 RESULTS

Herein the results of application of the tree construction and the succeeding refinement of reduced basis are presented. The snapshot matrix of observations for time function (See chapter 4.4.5) is processed. The computational complexity of the Finite Element Model equipped with Viscoplastic constitutive model is tackled using the refinement technique and the accuracy of the ROM is measured in both the offline and the online stages.

## 5.1 TIME FUNCTION

Data collected in a snapshot matrix for a certain time function (4.10) is processed by means of normalization (see Chapter 4.3). The comparison of the normalized and the original data is the base of discussion on feasibility of the refinement method.

The refinement methodology itself consists in procedures on the observed data collected into snapshot matrix. It starts from the tree construction (see Chapter 4.4) which is based on correlation between degrees of freedom. Then the base splitting is done according to the tree structure. Finally, the error has to be computed, in order to know the accuracy of chosen combination of branch of the tree and bases vectors.

In the beginning the refinement technique is applied to the normalized dataset as it was described in the Chapter 4.3. Figure 5.1 shows the resulting offline error (5.5) of Reduced Order Model at each step of basis refinement. It illustrates the fact that the use of normalized dataset in Figure 5.1 (right) gives the reduction of projection error with respect to original dataset in Figure 5.1 (left).

Projection error for the first basis that is split in 3 branches does not show better results than adding new bases (selection of 3 bases vectors for POD). Nevertheless, in Figure 5.1 (right) the error decreases 2 orders of magnitude for normalized dataset. For 3 basis vectors, after split in 2 branches the dimension becomes 6, which corresponds to FOM, so the relative error (4.21) between ROM and FOM drops to numerical zero.

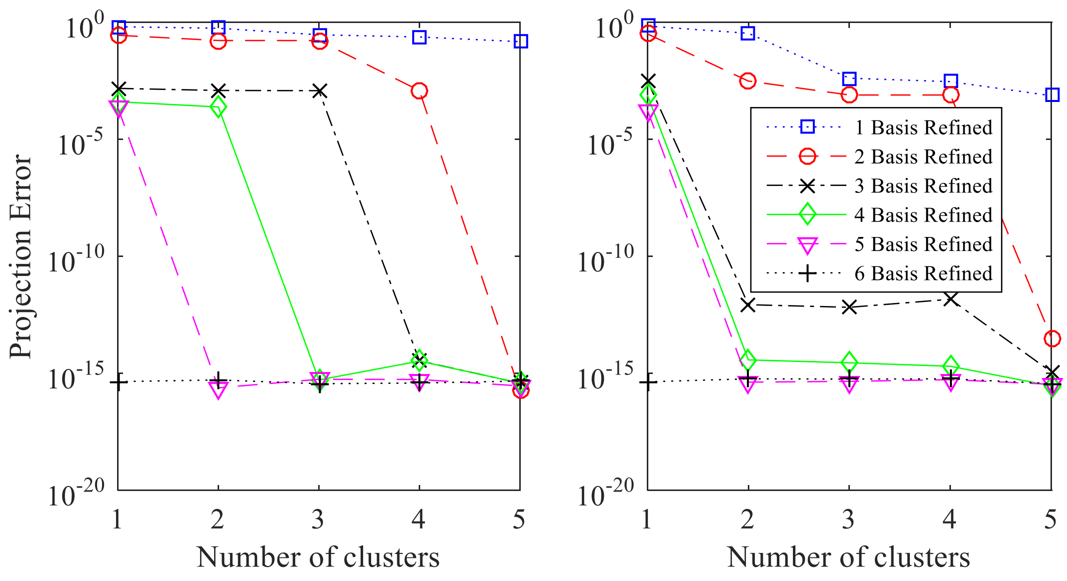


***Figure 5.1.*** *A-priori (projection) error. Time function*

*Not normalized (left) and normalized (right) observations.*

This example illustrates the fact that in the offline stage the projection error reduces if the dataset is normalized. It is important to keep in mind that the reduction is obtained by capturing the correlation between degrees of freedom. The a-priori error is defined by projection of the vector spanning in the reduced space onto the full order space. Due to the fact that the tree constructs sequence of hierarchical subspaces, h-refinement guarantees the convergence of ROM to FOM at the fine basis. Such a basis is a fully-split left orthogonal vector which has a dimension $N_{DOF} \times N_{DOF}$ after RRQR and normalization, and number of basis chosen times full depth of the tree $\times$ $N_{DOF}$.

| | Size of fully split (fine basis) | | | |
|---|---|---|---|---|
| Before RRQR | $N_{basis\ chosen} \cdot Depth\ of\ tree$ | $\times$ | $N_{DOF}$ | (5.1) |
| After RRQR | $N_{DOF}$ | $\times$ | $N_{DOF}$ | |

## 5.2 BAR UNDER TENSION

In this example the one-dimensional Finite Element Model equipped with Viscoplastic constitutive model is investigated. The non-linear bar under tensile forces has geometrical imperfections. The initial setup The geometry of the bar containing one imperfection is depicted at Figure 5.2 and cross-section area is defined at the Table 5.2. The Dirichlet boundary condition is applied at the constrained side of the bar and tensile force (Neumann boundary) is applied at the other end corresponding to (5.2) and Table 5.3. Viscoplastic constitutive model corresponds to Perzyna's formulation (See Chapter 3.1).

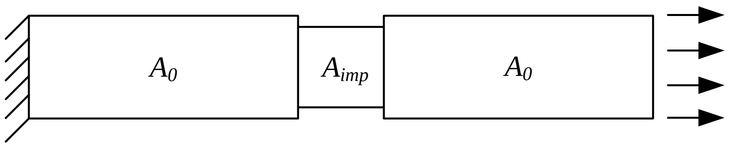


***Figure 5.2.*** *Geometry of the bar*

$$\begin{aligned} u(x)\big|_{x=0} &= 0 \\ \sigma(x)\cdot A(x)\big|_{x=L} &= F_{ext} \end{aligned} \tag{5.2}$$

The full order model that is solved by means of Newton-Raphson scheme (see Algorithm 1) and uses the programme setup from Table 5.1.

***Table 5.1 .*** *Model setup: G-FEM parameters. Strain-softening viscoplasticity.*

| **Definition** | **Variable** | **Value** | |
|---|---|---|---|
| **Number of time intervals** | model.nT | 100 | [s] |
| **End time** | model.T | 20 | [s] |
| **Time increment** | model.deltaT | 5 | [s] |
| **Length of the bar** | model.L | 100 | [m] |
| **Length imperfection zone** | model.impL | 5 | [m] |
| **Number of elements** | model.nel | 100 | [-] |

| | | | |
|---|---|---|---|
| **Max number of iterations per N.R. increment** | model.niter | 10 | [-] |
| **Convergence tolerance of N.R.** | model.tol | $10^{-6}$ | [-] |

The model parameters for inserting into the constitutive model of Perzyna viscoplasticity (see Chapter 3.1) are listed in the Table 5.2. Substituting the values into strain-softening response relation, it yields to

$$\sigma_Y = e^{-150\lambda} \tag{5.3}$$

where $\sigma_{Y0}$ is the initial yield stress, and $\lambda$ is the plastic strain and it behaves as depicted in Figure 5.3

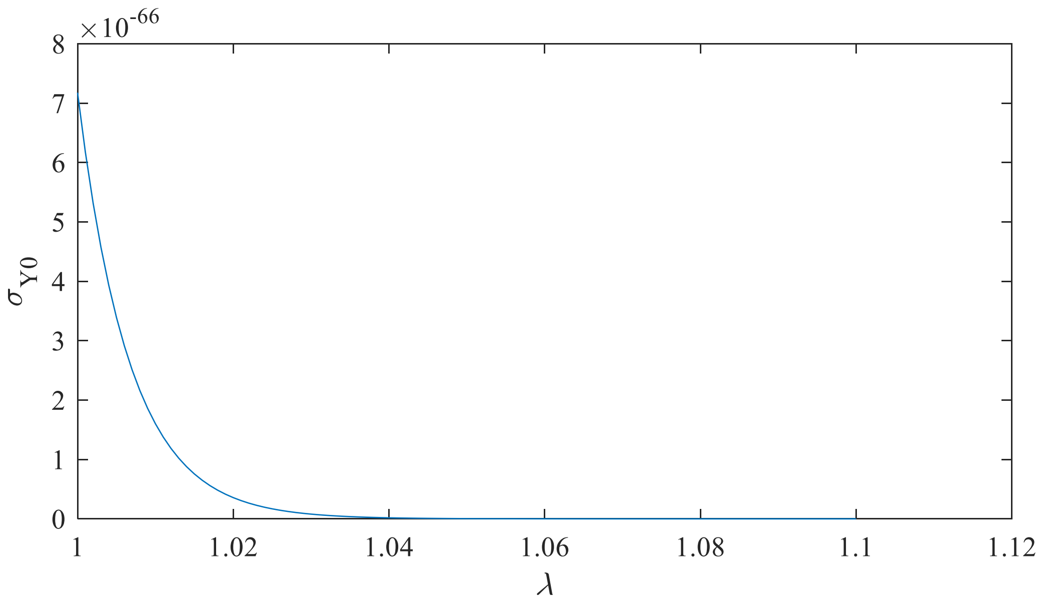


***Figure 5.3.*** *Strain-softening response curve*

***Table 5.2 .*** *Model setup: material parameters. Strain-softening viscoplasticity.*

| **Definition** | **Variable** | **Value** | |
|---|---|---|---|
| **Modulus of elasticity** | mat{1}.E | 1000 | [N / m$^2$] |
| **Yield stress** | mat{1}.Y | 1 | [N / m$^2$] |
| **Viscosity value Perzyna model** | mat{1}.eta | 10$^{-2}$ | [-] |
| **Exponent value Perzyna model** | mat{1}.N | 1 | [-] |
| **Hardening parameter Perzyna model** | mat{1}.Ya | -1 | [-] |
| **Hardening exponent parameter Perzyna model** | mat{1}.Yb | 75 | [-] |
| **Specimen area** | mat{1}.area | 1 | [m$^2$] |
| **Imperfection 1 area** | mat{2}.area | 0.89 | [m$^2$] |
| **Imperfection 2 area** | mat{3}.area | 0.90 | [m$^2$] |
| **Imperfection 3 area** | mat{4}.area | 0.91 | [-] |

In fact, the model with only one imperfection does not have enough contributing singular values. In order to discover the potential of MOR with h-refinement algorithm, three imperfections were introduced. The areas of imperfections are

$$\begin{array}{rl} A_0 & 0.00 < x < 0.22l \\ A^{I}_{imp} = 0.89A_0 & 0.22l < x < 0.27l \\ A_0 & 0.27l < x < 0.47l \\ A^{II}_{imp} = 0.90A_0 & 0.47l < x < 0.52l \\ A_0 & 0.52l < x < 0.72l \\ A^{III}_{imp} = 0.91A_0 & 0.72l < x < 0.77l \\ A_0 & 0.77l < x < 1.00l \end{array} \tag{5.4}$$

with the geometry as it is shown in Figure 5.4.

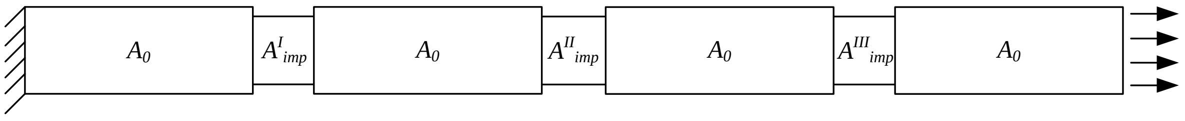


***Figure 5.4.*** *Geometry of the bar with three imperfections*

The boundary conditions remain to be (5.2)

***Table 5.3 .*** *Model setup: boundary conditions. Strain-softening viscoplasticity.*

| **Definition** | **Variable** | **Value** | |
|---|---|---|---|
| **Location of boundary** | bc.dirichlet.x | [0 model.L] | [m] |
| **Value of displacement at the boundary** | bc.dirichlet.ux | [0 1] | [m] |
| **Location of the applied force** | bc.neumann.x | model.L | [m] |
| **Value of the applied force** | bc.neumann.fx | 0 | [N] |

The displacement of the tip versus the applied force is plotted in Figure 5.5. At the initial loading stage the specimen behaves in elastic way, following a linear loading path. After passing the yielding point it reaches the peak value and due to strong localization of the stresses at the points of imperfection the strain-softening occurs. At the end of the loading stage when the value of displacement reaches 1, the residual is 0.25.

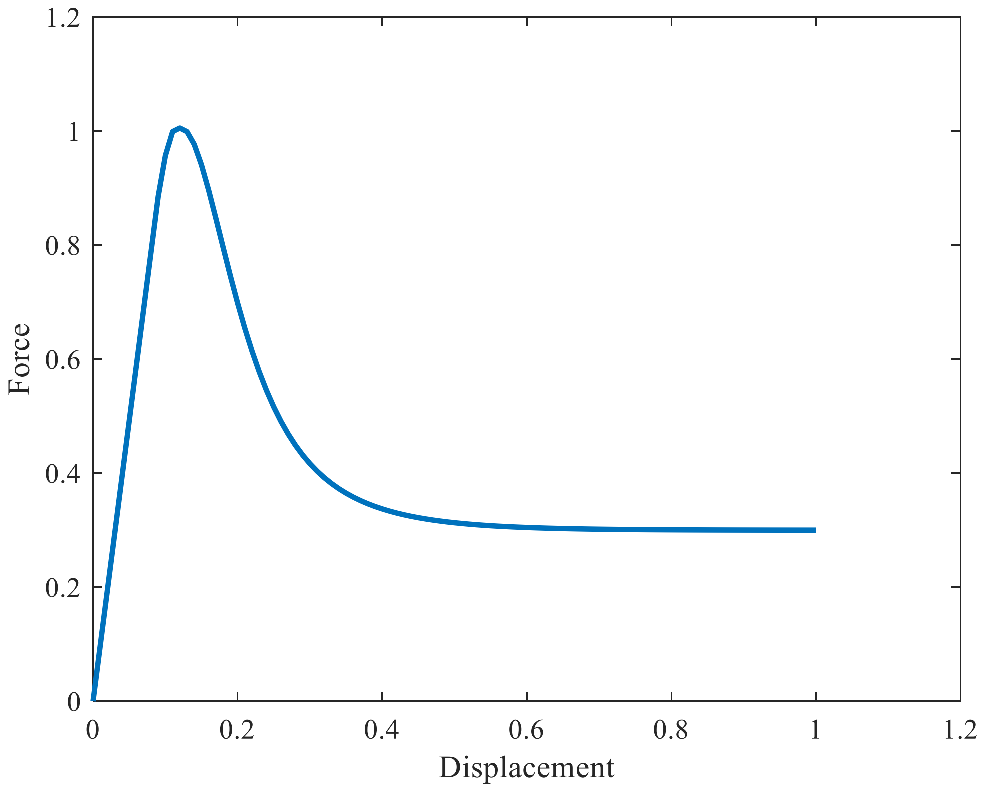


***Figure 5.5.*** *Force-displacement curve of the tip.*

The data from FOM results for all degrees of freedom from Figure 5.5 is collected into the snapshot matrix ***X***, where columns correspond to the taken snapshots, while rows contain the degrees of freedom.

Processing the snapshot matrix through the singular value decomposition (see Chapter 4.5) allows to plot the singular values in Figure 5.6. It is seen that four singular values are significantly contributing to the results of MOR.

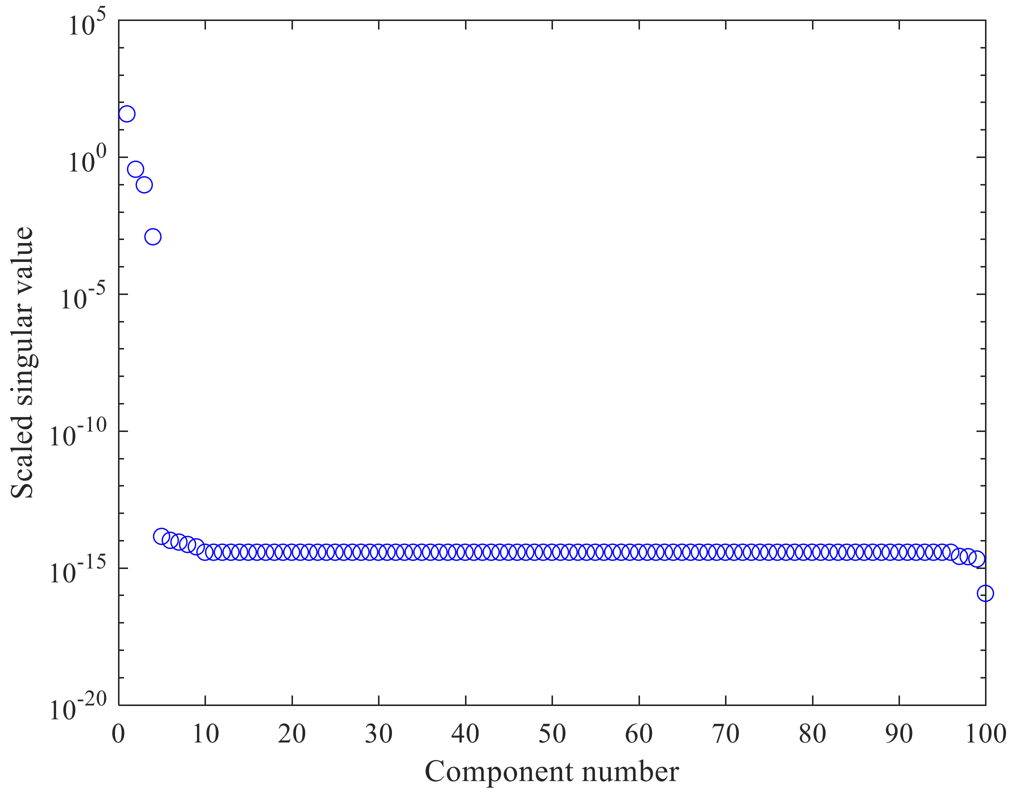


***Figure 5.6.** Singular values of the snapshot matrix. Viscoplasticity.*

The basis vectors of this snapshot matrix could be described by four values of SVD, which means that taking four basis is sufficient for full description of this problem in a framework of ROM.

Snapshot matrix $\boldsymbol{X}$ is not processed, which means that no normalization from Algorithm 3 was done. For this original dataset the tree construction and refinement shows that the projection error in offline stage gradually decreases as the number of basis vectors increase. In Figure 5.7 (left) the projection error of the refinement of basis vector with not corresponding tree are presented. It could be told that the tree structure does not correspond to the snapshot matrix when, for instance, as in this case normalized dataset is used for the construction of the tree, while the matrix of observations $\boldsymbol{X}$ remains not-normalized. As it was illustrated before in the Figure 5.1, normalization of observations provides faster a-priori error reduction. In the Figure 5.7 (right) the tree fully corresponds to the non-normalized dataset.

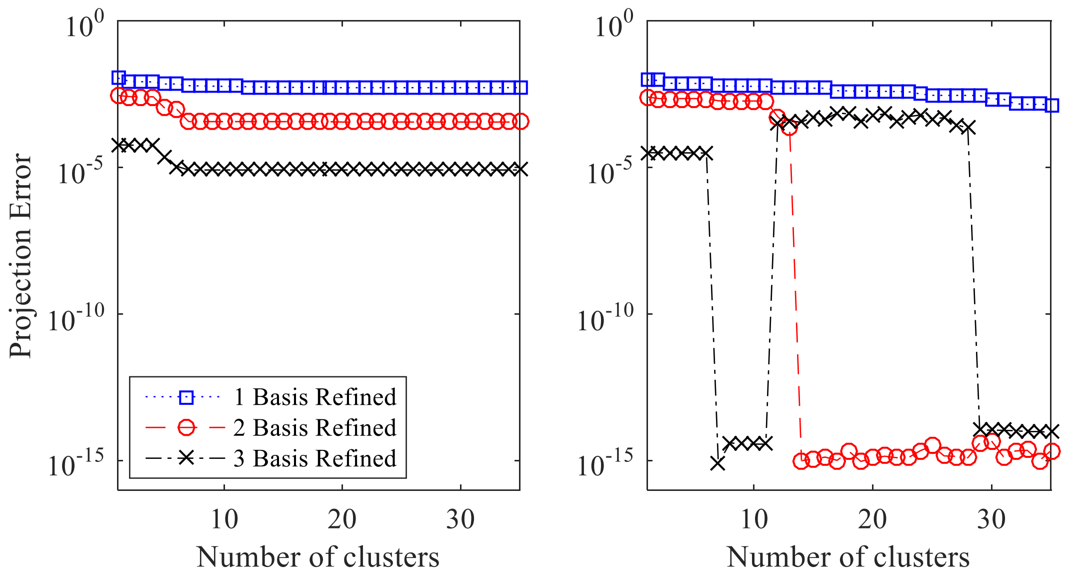


***Figure 5.7.*** *A-Priori (projection) error. Viscoplasticity*

*Not corresponding tree (left) and not normalized observations (right)*

The projection error calculated a-priori has a trend to decrease by refining basis vectors. In this example four significant bases vectors are present, meaning that four singular values σ are nonzero.

The projection error

$$\frac{\left\| \boldsymbol{X} - \boldsymbol{V}^{h} (\boldsymbol{V}^{h})^{\dagger} \boldsymbol{X} \right\|_F}{\left\| \boldsymbol{X} \right\|_F} \tag{5.5}$$

where † denotes Moore–Penrose pseudoinverse and $\left\| \boldsymbol{X} \right\|_F$ is a norm defined by Equation (4.6), is shown in the Figure 5.7. Choosing just one basis for refinement, the error gradually decreases, but the refinement does not introduce significant error drop. While for 2 basis vectors the significant drop occurs when the branch is equal to 14. Three basis reach the error reduction faster when the chosen branch is 7. In both cases of splitting the basis 2 or 3 the projection error reaches value of using 4 basis vectors. It is seen that splitting the basis does not produce improvement with respect to adding the basis vector. Nevertheless, the produced result show that the desired value of error could be reached without adding the basis vectors, but by means of splitting the reduced basis. This happens when size of $\boldsymbol{V}^{H}$ tends to $N_{DOF} \times N_{DOF}$ and described by the property 2 of the tree (see Chapter 4.4.6).

Observations derived from the described behaviour of projection error could be summarized in two facts. First of all, the importance of using the proper tree corresponding to snapshot matrix on a refined basis $\boldsymbol{V}^h$ is shown by comparison of left and right plots of Figure 5.7. The meaning of 'proper tree' in this context is that the tree was constructed on the basis of the same snapshot matrix $\boldsymbol{X}$, as the basis $\boldsymbol{V}^h$ was computed. Secondly, the right plot demonstrates that the error reduction could be achieved by means of refinement. Also, it is important to mention that the process of refinement should be terminated at a certain point, in order to avoid the error increase, which occurs when the columns of $\boldsymbol{V}^h$ are linearly dependant.

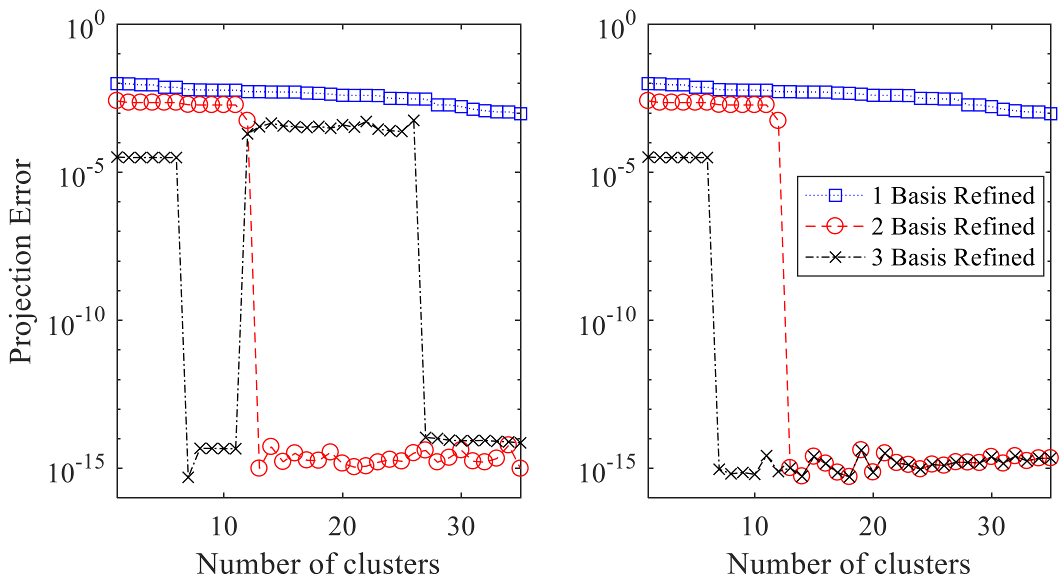


***Figure 5.8.*** *A-Priori (projection) error. Viscoplasticity*

*Not normalized observations (left) and RRQR not normalized observations (right)*

According to the properties of the tree described in Chapter 4.4.6, the refinement provides the creation of hierarchical subspaces $range(\boldsymbol{V}^H) \subseteq range(\boldsymbol{V}^h)$. So, conceptually the projection error has to decrease monotonically. Due to the fact that during the refinement the basis $\boldsymbol{V}^H$ is split, columns of $\boldsymbol{V}^h$ might become linearly dependent. In order to avoid the phenomena appearing in Figure 5.8 (left) for case of 3 basis, split 12, it is needed to remove linear dependence of columns of the refined left orthogonal vector $\boldsymbol{V}^h$. Rank-Revealing QR factorization (RRQR) is aimed to do so (see Chapter 4.6). The avoidance of singularity of $\boldsymbol{V}^h$, when condition number is high provides results of Figure 5.8 (right). The error does not only remain stable, but also the size of $\boldsymbol{V}^h$ is shrunk by use of RRQR, which leads to further reduction of the computational cost.

The relative error

$$\frac{1}{t}\sum_{k=1}^{t}\frac{\left\| u_{FOM}(\cdot,\tau^k) - u_{ROM}(\cdot,\tau^k)\right\|_{L^2}}{\left\| u_{FOM}(\cdot,\tau^k)\right\|_{L^2}} \tag{5.6}$$

is computed during online stage. It decreases by refining basis vectors, as it is shown in the Figure 5.5 (left). There could be seen a correlation between a-priori and a-posteriori error. During the offline stage the accuracy of refinement is defined by projection error calculated with Equation (5.5).

The accuracy of online stage is measured by calculation of residual (ROM solution extracted from FOM solution) and normalized with respect to the result of full order solution. Obviously, during the real online stage the use of Equation (5.6) is limited, since there is no possibility to know FOM solution. Nevertheless, since for this illustrative example FOM solution is provided, the relative error can be computed a-posteriori.

Figure 5.5 (right) shows the fact that using one basis the error gradually decreases, but never reaches the low value of error in a certain span of [1; 35] basis refinements. The use of two basis vectors shows significant error drop at 14 branches and then a further drop to desired value at 22 branches, when the size of the basis reaches 22, thanks to RRQR. The first drop in the case of 3 basis also occurs for 3 branches in both cases of a-priori and a-posteriori errors and at a certain point when the size of the basis is equal to 19, for 2 and 3 basis chosen the relative error starts to follow the same path. From the projection error is hard to predict the occurrence of the second error drop at a branch 22 with size of the basis 29, when the relative error reaches desired value.

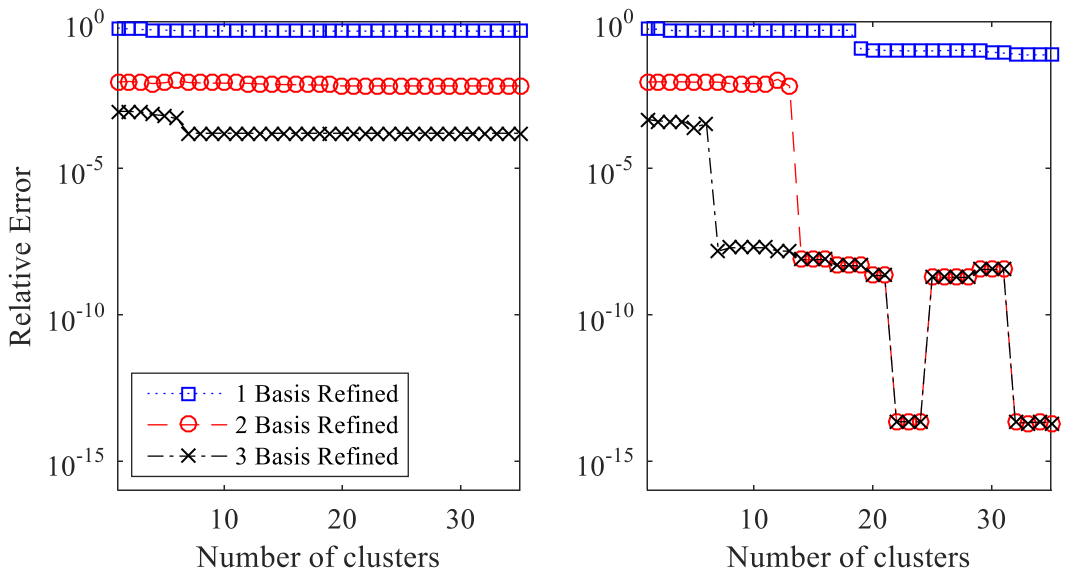


***Figure 5.9.*** *A-Posteriori (relative) error. Viscoplasticity*

*Not corresponding tree (left) and not normalized observations (right)*

Basically, the same observation of the necessity of correspondence of the tree to the snapshot matrix is induced from comparison of plots from Figure 5.9. The trend of the error reduction by means of refinement is observed on the right plot. The exact point of termination of refinement process has to be investigated in details. Also the comparison with Figure 5.7 shows that a-posteriori error is hard to predict during online stage.

In order to see the evolution of the relative error during the process of basis refinement, it is necessary to plot the relative error and the size of the basis simultaneously on the same canvas. In Figure 5.10 the left scale corresponds to the computed relative error, the right scale shows size of the basis during refinement procedure.

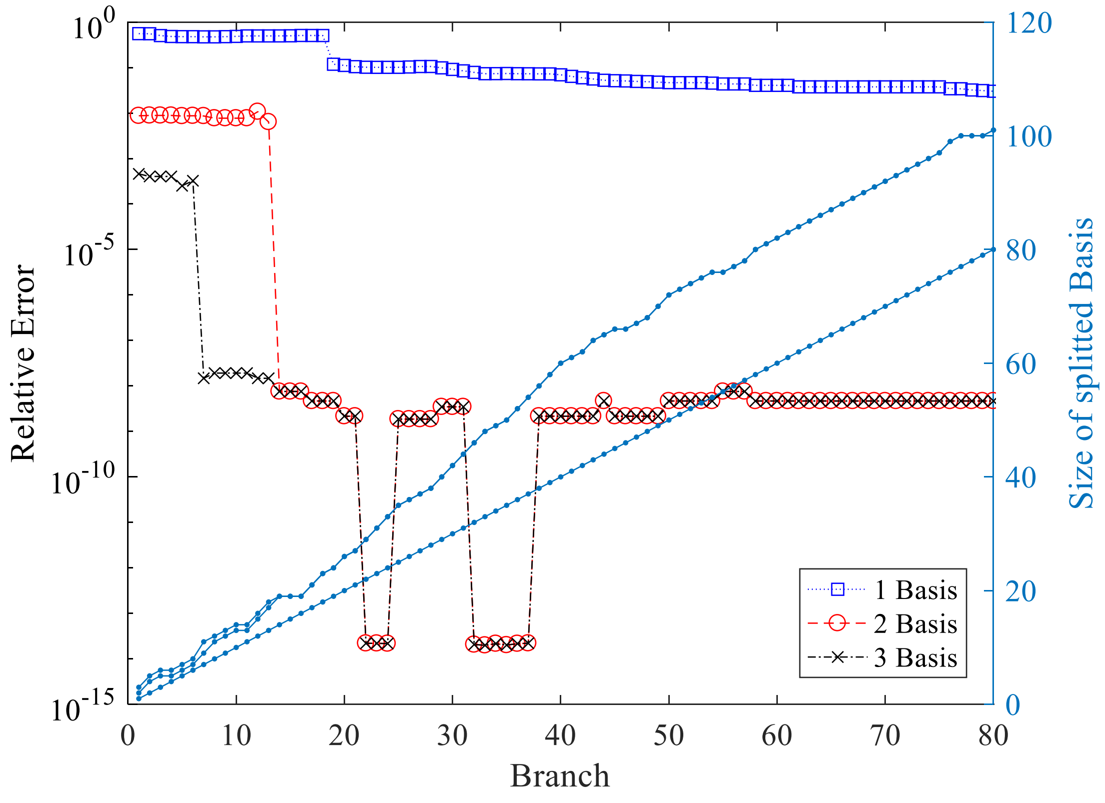


***Figure 5.10.*** *A-posteriori relative error for not normalized observations(left scale) and size of splitted basis (right scale). Viscoplasticity*

Taking initially one basis vector as $\boldsymbol{V}^{H}$ the relative error does not drop, and the size of the basis increases linearly. It is possible to predict that the error drop would occur due to the property 2 of the tree (see Chapter 4.4.6), but it is less efficient then adding another basis to $\boldsymbol{V}^{H}$. In the case of choosing two or three basis vector as $\boldsymbol{V}^{H}$ the value of error becomes similar when both basis have the same size. Even thought the size of the basis $\boldsymbol{V}^{h}$ has to increase dramatically at size($\boldsymbol{V}^{H}$)˙ split, it does not happen due to he RRQR

## 5.3 INVISCID BURGERS EQUATION

### 5.3.1 *FULL ORDER MODEL*

The setup used for the initial solution is parameterized boundary value problem

$$\frac{\partial u\ (x,\tau)}{\partial \tau}+\frac{1}{2}\frac{\partial\left(u^2\ (x,\tau)\right)}{\partial x}=0.02\,e^{\mu_2}x \tag{5.7}$$

with boundary conditions

$$u\,(0,\tau)=\mu_1\,,\ \forall\,\tau>0 \tag{5.8}$$

$$u\,(x,0)=1\,,\ \ \forall x\in[0,\,100] \tag{5.9}$$

where $\mu_1$ and $\mu_2$ are two real-valued input variables is defined by Table 5.4.

In the time interval $\tau\in[0,\,50]$ the solution $u\,(x,\tau)$ is computed using a uniform computational time-step size $\Delta t$ =0.05, leading to $t$ = 1000 total time steps. During the offline stage, snapshots of the state are collected into the snapshot matrix $X$ for the time steps at training inputs. Newton-Raphson scheme with a tolerance $10^{-8}$ is used at a full order model (see Table 5.5).

***Table 5.4 .*** *Model setup. Burgers Equation*

| **Definition** | **Variable** | **Value** | |
|---|---|---|---|
| **Number of time intervals** | model.nT | 1000 | [s] |
| **End time** | model.T | 50 | [s] |
| **Time increment** | model.deltaT | 0.05 | [s] |
| **Length of the bar** | model.L | 100 | [m] |
| **Length imperfection zone** | model.impL | 10 | [m] |

| | | | |
|---|---|---|---|
| **Number of elements** | model.nel | 150 | [-] |
| **Max number of iterations per N.R. increment** | model.niter | 10 | [-] |
| **Convergence tolerance of N.R.** | model.tol | $10^{-8}$ | [-] |

In this case the produced solution is not smooth as it is depicted in the Figure 5.11. The G-FEM with the Newton-Raphson scheme fails to capture the shock wave formation, so the oscillations in the field of *u* occur.

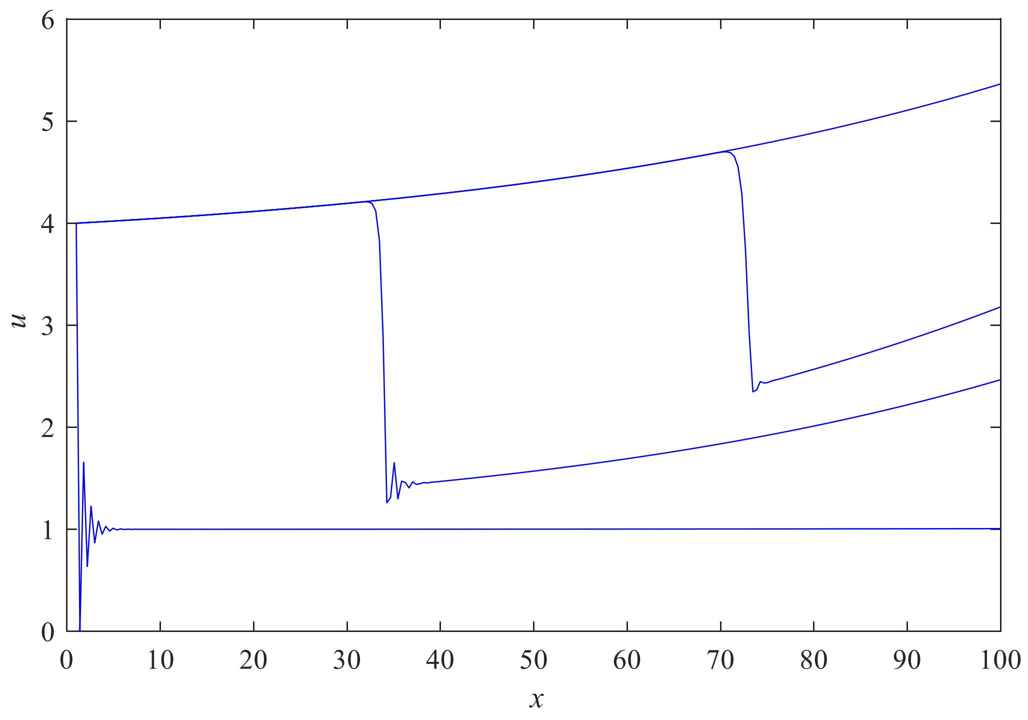


***Figure 5.11.*** *Full Order Model discretized in 150 elements. Inviscid Burgers equation.*

Increasing the number of elements model.nel = 1000 and leaving all the rest of the input parameters from Table 5.4 the result becomes much smoother, as it is depicted at Figure 5.12.

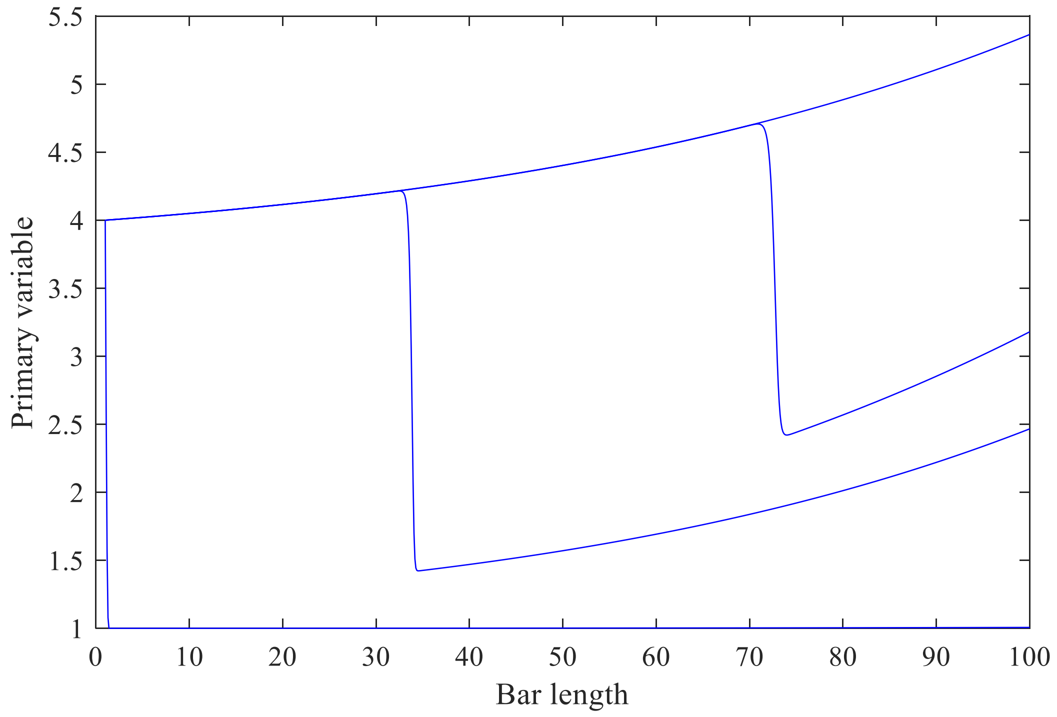


***Figure 5.12.*** *Full Order Model discretized in 1000 elements. Inviscid Burgers equation.*

In such a case the dimension of the model has drastically increased from 150 to 1000, as well as the solution of the linearized system of equations became more computationally expensive. The singular value coefficients on the Figure 5.13 have significant number of non-zero values. Moreover a slow decay of the plot illustrates the fact of high nonlinearity of the model based on Inviscid Burgers equation.

The offline stage of the ROM procedure includes computation of the left orthogonal basis $V^H$ based in the SVD procedure. The tree construction algorithm, described at the Chapter 4.4 is employed for the offline stage. The constructed tree based on the snapshot matrix $\boldsymbol{X}$ can be seen as a dendrogram of Figure 5.14, according to the structure of hierarchical binary clustering (see Chapter 4.4.2). Some degrees of freedom are separated by a high distance calculated with a metric of Euclidean distance (See Chapter 4.4.4). The rest have a small distance, which is good sign for defining the groups, e.g. branches for basis splitting. The created tree structure is stored for use on the online stage in the array consisting of the branches.

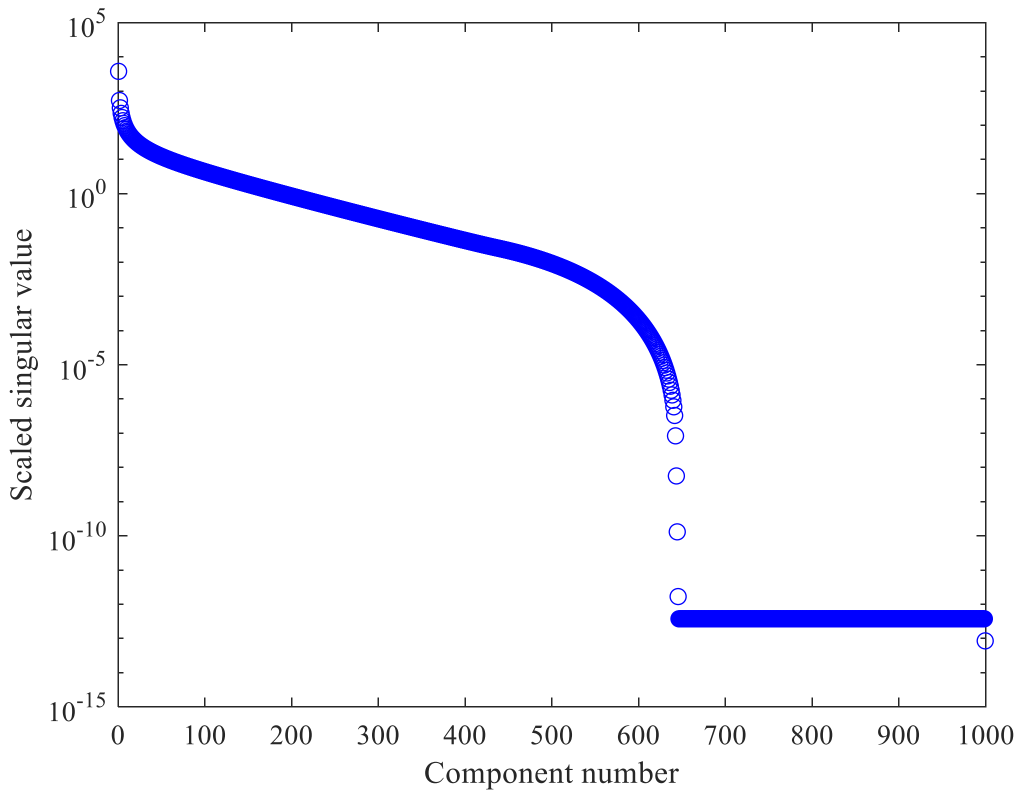


***Figure 5.13.*** *Singular values of the snapshot matrix. Inviscid Burgers equation.*

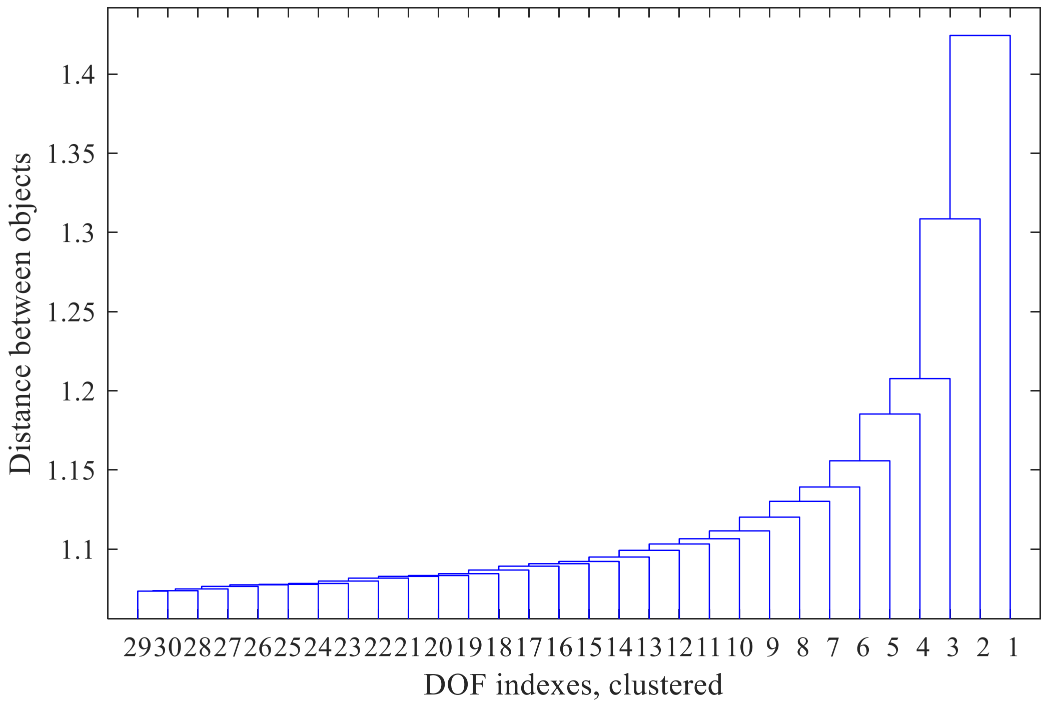


***Figure 5.14.*** *Dendrogram. Inviscid Burgers equation.*

### 5.3.2 *Reduced Order Model*

In the offline stage is possible to compute the projection error defined in Chapter 5.3.3. The projection error plotted on the Figure 5.15 gradually decreases, as long as the basis vector $\boldsymbol{V}^H$ is refined according to the tree structure. The values of offline error do not decrease dramatically adding the basis vectors (increasing the size of left orthogonal vector by adding basis). Just a slight reduction occurs taking the basis from 50 to 250, with a step of 50, e.g the basis size is higher 5 times than the initially taken.

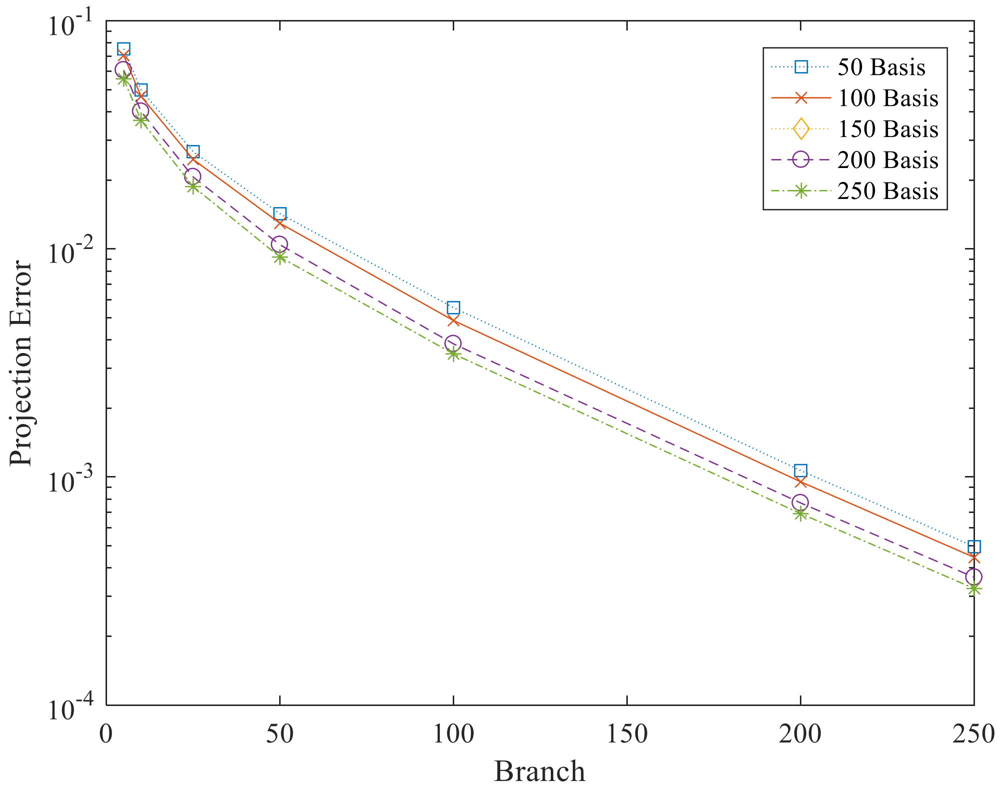
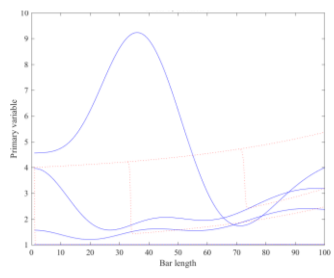
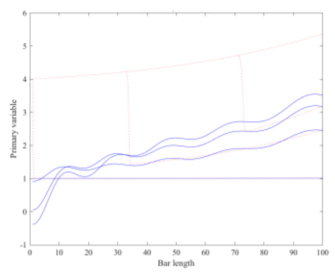
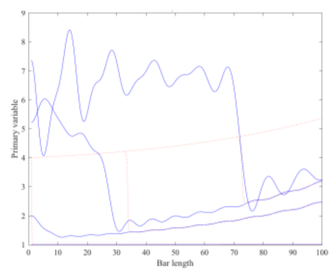
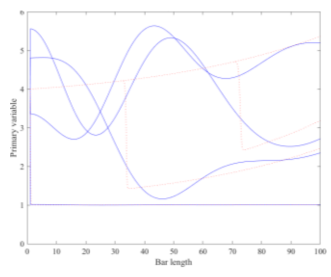
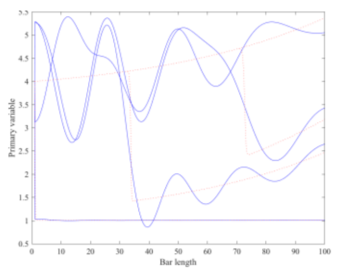
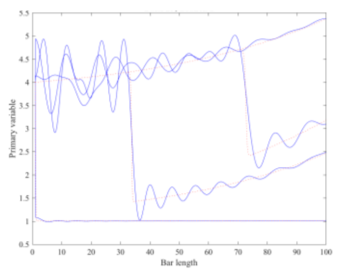
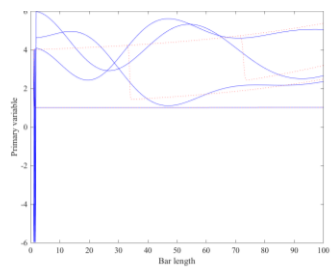
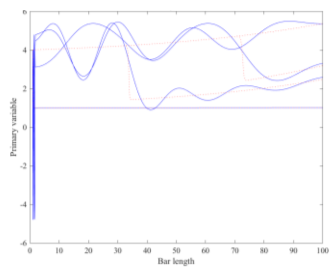


***Figure 5.15.*** *Projection Error. Inviscid Burgers equation.*

In order to check the trend of error reduction observed a-priori, it is necessary to execute the online stage and compare the results of ROM and FOM. In the Table 5.5 the ROM results (solid line) of taking 5, 10 and 25 basis vectors are compared to the Full order solution (dashed line) in the first row. It is clearly seen that the ROM fails to reproduce the solution of Burgers Equation. The same happens for splitting the basis 5 and 10 up to 10 branches. In the case of splitting the left orthogonal vector $\boldsymbol{V}^H$ of the size 25 once (into 2 branches) the resulting picture slightly resembles the captured behaviour of FOM, nevertheless the resulting error computed offline is 10%. It is still not acceptable result for the ROM performance.

***Table 5.5.*** *ROM basis splitting into branches. Burgers Equation*

| **Branch** | **Basis 5** | **Basis 10** | **Basis 25** |
|---|---|---|---|
| **1** |  |  |  |
| **2** |  |  |  |
| **10** |  |  | |

It was observed in the example of the bar under tension in Chapter 5.2 that the basis splitting reduces the relative error. Tests on the ROM without basis splitting (1 Branch) for 50 and 200 basis show that ROM alone cannot reach the FOM solution.

Increasing the size of the basis by means of h-refinement from 50 basis to 100 splitting into two branches does not mean that the size of $\boldsymbol{V}^h$ remains the same. After processing $\boldsymbol{V}^h$ trough RRQR the linear dependency of columns becomes eliminated. In this case the size of the basis changes from 50 to 51, but the offline error drops from 56% to 3,6%. It also seen on the Figure 5.16 that the further splitting into 10 branches does not reduce the error significantly. It is important to keep in mind that instead of the Full Order Model requiring the solution of the system of size 1000, the system solved here has dimensionality of 51, resulting in 3,6% error.

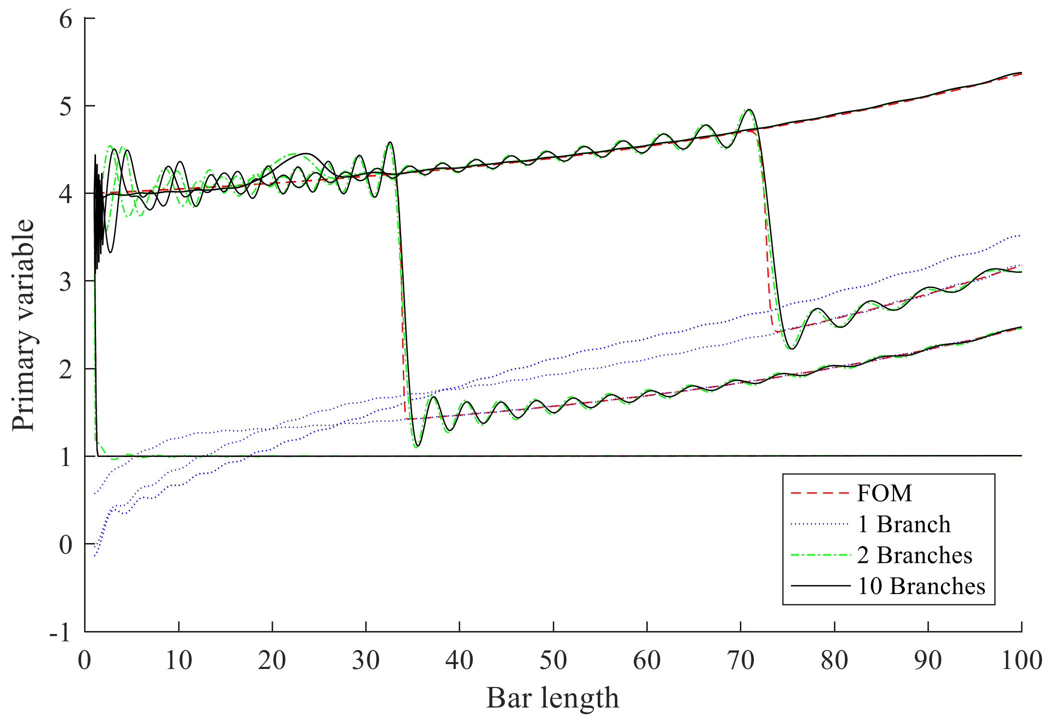


***Figure 5.16.*** *ROM with 50 basis vectors. Inviscid Burgers equation.*

For illustrative purposes, it is possible to take 200 basis vectors, and split in the same fashion as before, producing the Figure 5.17. It results in error drop from 45% to 0,2% as at solving a five times smaller system as the initial FOM.

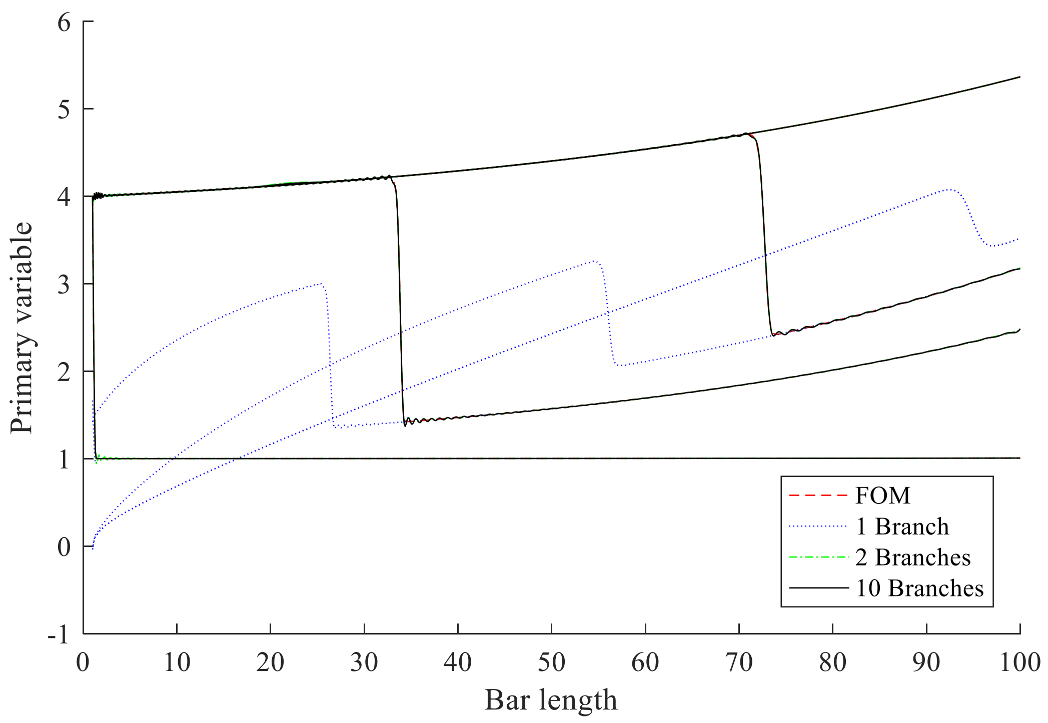


***Figure 5.17.*** *ROM with 200 basis vectors. Inviscid Burgers equation.*

### 5.3.3 *Error computation*

For the available model of Burgers equation the error reduction occurs after a first basis splitting. For the number of branches greater than two, the error does not decrease significantly (see Figure 5.18).

Adding the basis to the left orthogonal vector and executing ROM alone shows incapability of capturing the phenomena. H-refinement coupled with adding the basis reduces the online error.

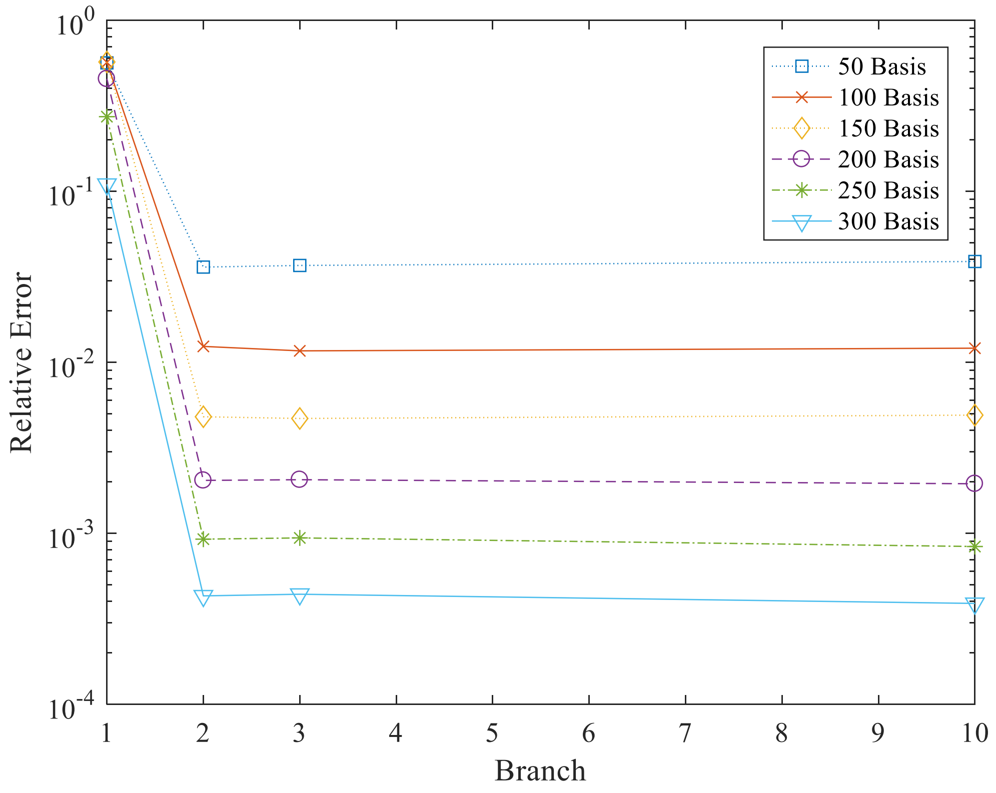


***Figure 5.18.*** *Relative error of ROM. Inviscid Burgers equation.*

Comparing trends of offline error (Figure 5.15) and error computed online (Figure 5.18) it is hard to define a-priori the proper choice of the basis and branch to be efficient online.

# 6 Conclusion and Future development

On the basis of the observations from the numerical results, it is possible to conclude that the reduction in projection error is observed for nonlinear problem. The Rank-Revealing QR-factorization helps in removing linear dependency of the columns of basis vector and as a consequence, prevents the increase of the projection error offline.

In the online stage, for the current viscoplastic model an error reduction occurs. The use of RRQR is mandatory here, since the system of equation cannot be solved without positive definite stiffness matrix. Nevertheless, the use of RRQR does not eliminate spurious increase of the relative error computed during the online stage.

Comparison of the h-refinement technique with the standard ROM, shows that one dimensional bar equipped with Perzyna viscoplastic model does not show advantage of the use of h-refinement (basis splitting). However in the case of the problem of Burgers equation, where the response can be discontinuous, and more singular values contribute to the solution of ROM (when the typical ROM basis is not truncated significantly), the advantage of using ROM with h-refinement is visible. It was illustrated that when POD alone fails to capture the shock formation, the use of refinement technique is able to deliver an accurate solution without a noticeable increase of the computational cost.

# 7 APPENDIX A. CODE MANUAL

The given code manual describes the workflow and operations of the MatLab code devoted to Model order reduction, h-refinement technique. It includes full order model solution for 1D bar, tree construction procedure and computation of the error in both: offline and online stages. The reduced order model is trained on the basis of the full order model (FOM), and the succeeding solution is obtained through model order reduction.

## 7.1 TREE CONSTRUCTION

The tree construction procedure is described by a flow chart at the end of this section in ***Flowchart 4***. The results of the solution of FOM is used as a training set for offline stage. The matrix of snapshots ***X*** has a format where rows contain degrees of freedom and columns the snapshots, and it has a dimensionality of $N_{Samp} \times N_{DOF}$.

### 7.1.1 *NORMALIZATION OF SNAPSHOTS*

In order to capture the correlation of the degrees of freedom, it is possible to normalize training set according to

$$\begin{aligned} x_{i,j} &\leftarrow \frac{x_{i,j}}{\left\| \underline{x}_i^T \right\|} \\ \underline{x}_i^T &\leftarrow -\underline{x}_i^T;\ if\ x_{i1} < 0 \end{aligned} \tag{7.1}$$

where $\|.\|$ denotes a $L^2$ norm

Through the normalization, the correlated and anticorrelated state variables become separated by small Euclidean distance and can be grouped via clustering.

### 7.1.2 *Binary clustering*

In order to proceed on clustering algorithm it is needed to establish a metric for the pairwise distance. The way of capturing similarities between observations in this case is the Euclidean distance computed by

$$d^2_{ij} = (\underline{x}_i - \underline{x}_j)(\underline{x}_j - \underline{x}_j)' \tag{7.2}$$

where for $\underline{x}_i$ and $\underline{x}_j$ are row vectors of snapshot matrix $\underline{\underline{X}} \in \mathbb{R}^{N_{Samp} \times N_{DOF}}$

There exists other ways of computation the distance, as for instance cosine distance evaluated for vectors $\underline{x}_i$ and $\underline{x}_j$ as

$$d_{ij}(\underline{x}_i, \underline{x}_j) = 1 - \cos(\underline{x}_i, \underline{x}_j) = 1 - \frac{\underline{x}_i \underline{x}_j{}'}{\sqrt{(\underline{x}_i \underline{x}_i{}')(\underline{x}_j \underline{x}_j{}')}} \tag{7.3}$$

It is observed in [13] that having large dimensionality of vectors, the difference between results of computation Euclidean or cosine distance is not significant.

Binary clustering is adopted for capturing the correlations between degrees of freedom. The Euclidean distance is computed at the each iteration, defining the metric for linking pairs of clusters. It creates the matrix $(2N_{DOF} - 1) \times 3$

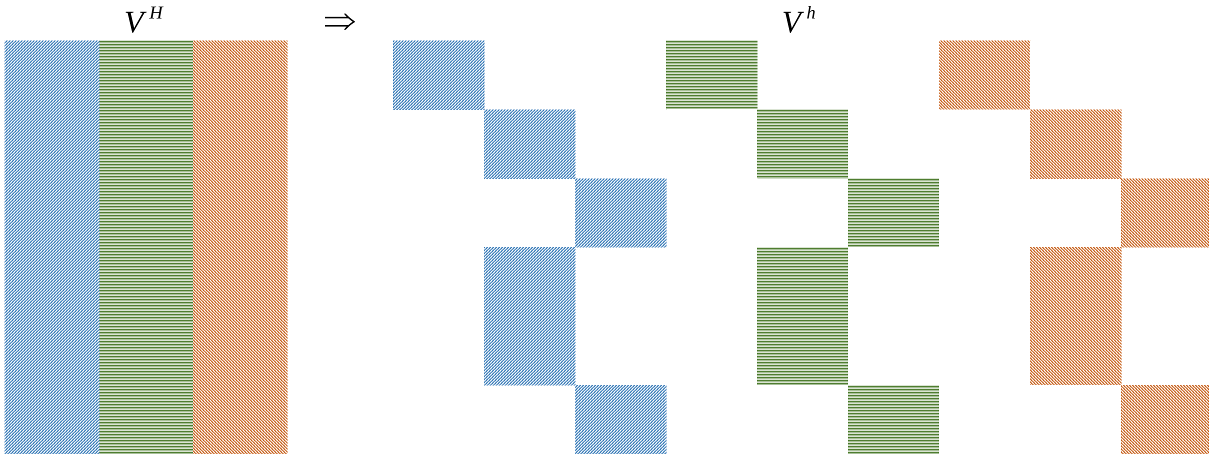
| Global Cluster | Left Cluster | Right Cluster | Distance |
|---|---|---|---|
| {1} | 1 | 1 | 0 |
| ⋮ | ⋮ | ⋮ | ⋮ |
| {6} | 6 | 6 | 0 |
| {7} | 4 | 5 | 0.0039 |
| {8} | 2 | 7 | 0.0062 |
| {9} | 3 | 6 | 0.0071 |
| {10} | 1 | 9 | 1.0083 |
| {11} | 8 | 10 | 1.3088 |

(7.4)

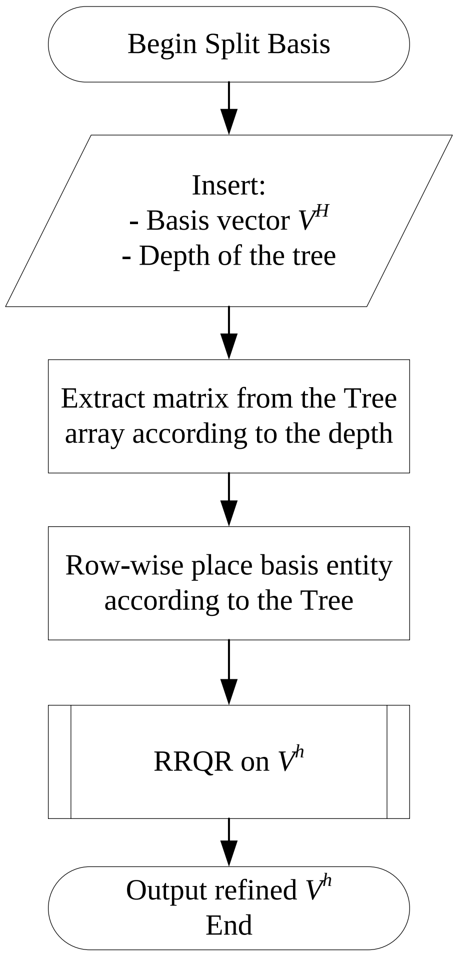
where the Global Cluster raw contains the left and right cluster, defining the linked pair. The last column defines the value of the metric which is used for creating a new cluster. At the next stage the names of the clusters are replaced with the clusters form *1* to $N_{DOF}$ themselves, such as Global Cluster {10}, containing global clusters {1} and {9} becomes {1, 3, 6} after substitution.

### 7.1.3 *Refinement Mechanism*

As soon as the branch and size of the basis $V^h$ is chosen, the depth of the tree is set. The corresponding level of the tree is chosen and the basis vector $V^H$ is split the following manner



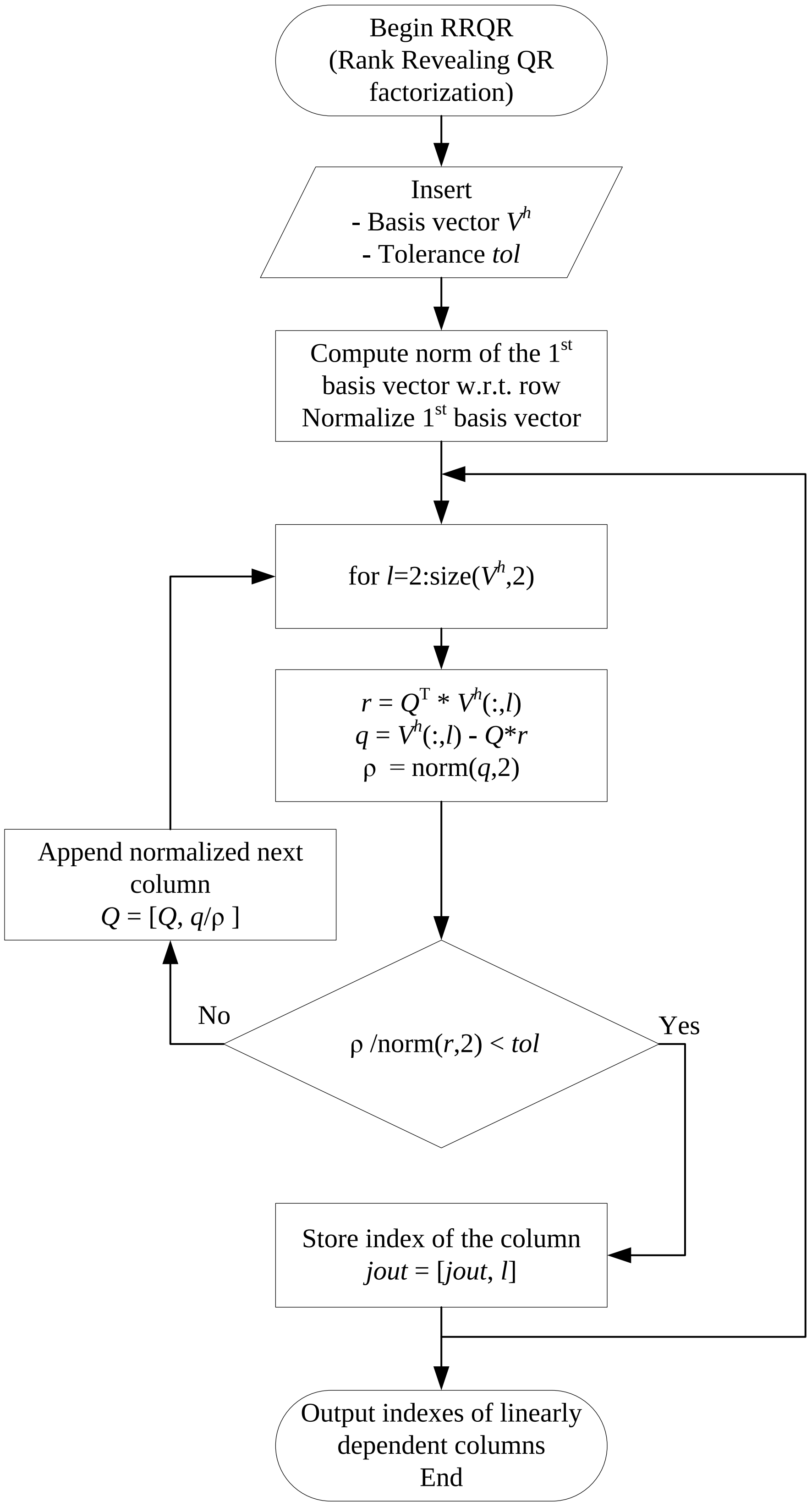
The “split basis” procedure places the entities of basis vector $V^h$ at the positions prescribed by the tree. The entire procedure is as described at the Flowchart 1


Begin Split Basis
Insert:
- Basis vector $V^H$
- Depth of the tree
Extract matrix from the Tree array according to the depth
Row-wise place basis entity according to the Tree
RRQR on $V^h$
Output refined $V^h$
End

***Flowchart 1.*** *Split basis*

### 7.1.4 *RANK REVEALING QR-FACTORIZATION*

In the case when it is needed, the refined basis vector $V^h$ could be processed though the rank-revelling QR factorization, in order to eliminate linear dependency of the columns Flowchart 2


Begin RRQR
(Rank Revealing QR factorization)
Insert
- Basis vector $V^h$
- Tolerance *tol*
Compute norm of the 1st basis vector w.r.t. row
Normalize 1st basis vector
for $l$=2:size($V^h$,2)
$r = Q^T * V^h(:,l)$
$q = V^h(:,l) - Q*r$
$\rho = \text{norm}(q,2)$
Append normalized next column
$Q = [Q, q/\rho]$
No
$\rho / \text{norm}(r,2) < tol$
Yes
Store index of the column
$jout = [jout, l]$
Output indexes of linearly dependent columns
End

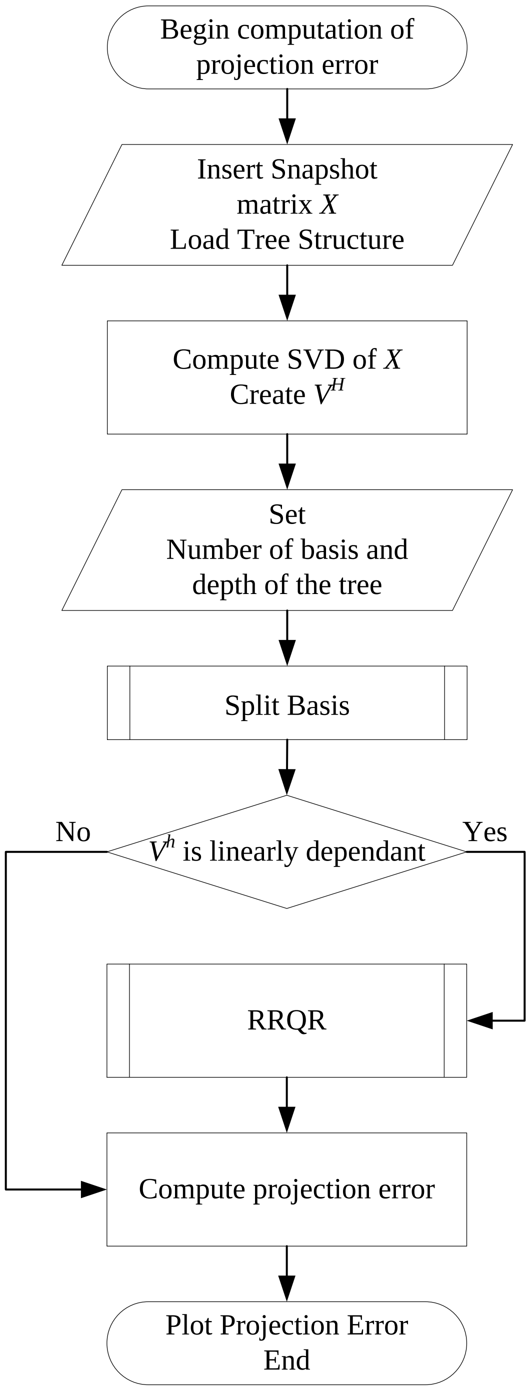
***Flowchart 2.*** *Rank Revealing QR-Factorization*

### 7.1.5 *Projection error*

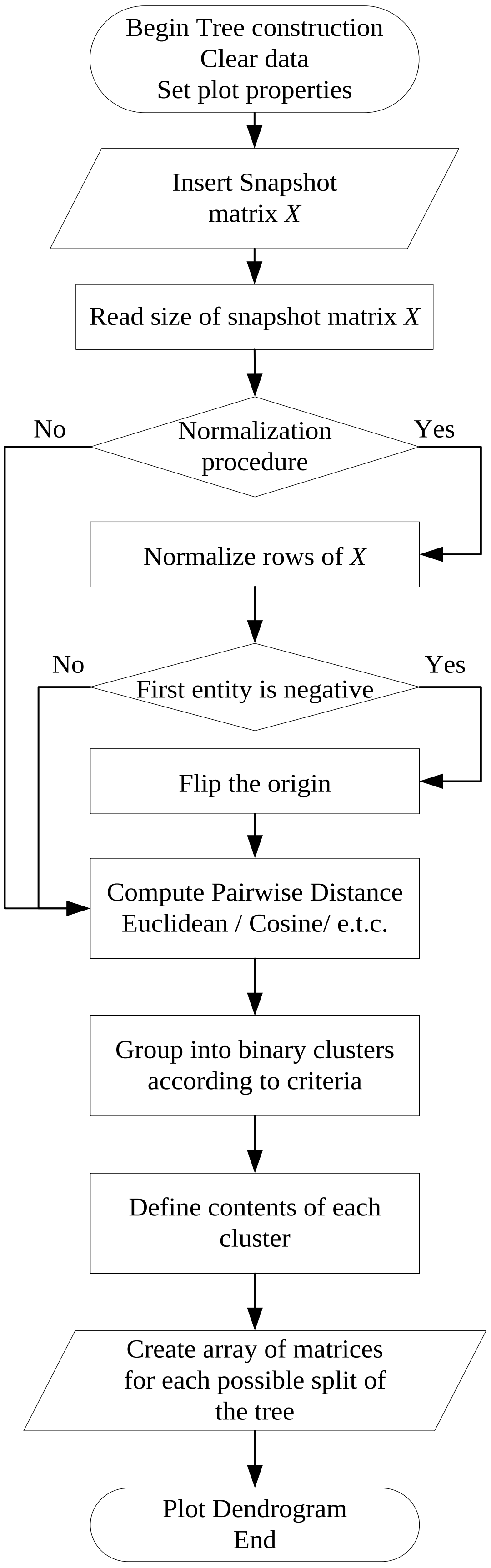
The projection error is computed for a certain branch and certain basis of the tree as

$$\frac{\left\| X - V^h (V^h)^{\dagger} X \right\|_F}{\|X\|_F} \tag{7.5}$$

where † denotes Moore–Penrose pseudoinverse and $\|X\|_F$ is a Frobenious norm. The procedure of computation of the offline error is described by Flowchart 3


Begin computation of projection error
Insert Snapshot matrix $X$ Load Tree Structure
Compute SVD of $X$ Create $V^H$
Set Number of basis and depth of the tree
Split Basis
No
$V^h$ is linearly dependant
Yes
RRQR
Compute projection error
Plot Projection Error End

***Flowchart 3.*** *Offline (projection) error*



Begin Tree construction
Clear data
Set plot properties
Insert Snapshot matrix X
Read size of snapshot matrix X
Normalization procedure
No
Yes
Normalize rows of X
First entity is negative
No
Yes
Flip the origin
Compute Pairwise Distance Euclidean / Cosine/ e.t.c.
Group into binary clusters according to criteria
Define contents of each cluster
Create array of matrices for each possible split of the tree
Plot Dendrogram
End

***Flowchart 4.*** *Tree construction*

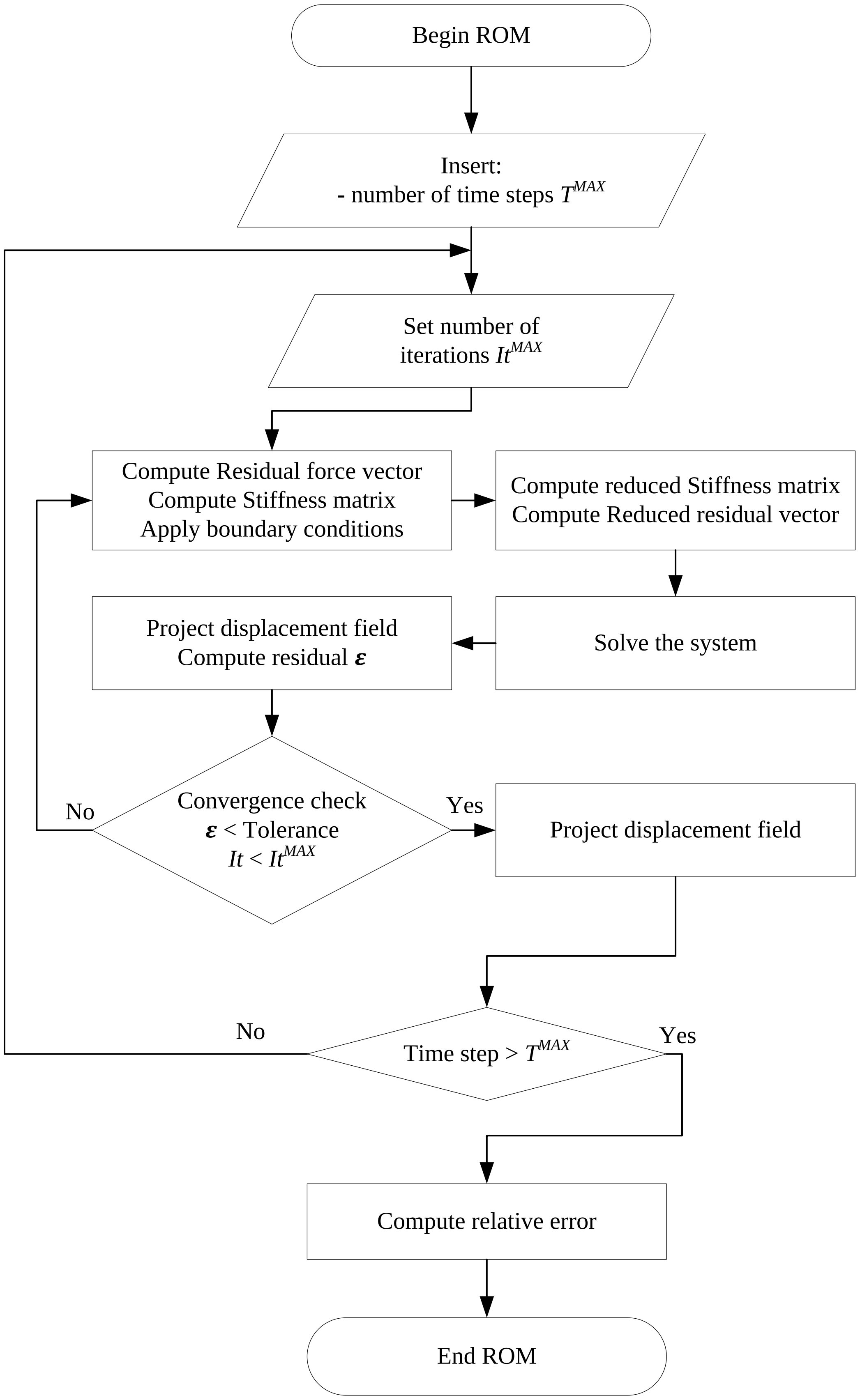
## 7.2 REDUCED ORDER MODEL

The traditional Newton-Raphson scheme is employed for the ROM as at is depicted at Flowchart 5.

The snapshot matrix $X$ is decomposed according to the SVD in order to create reduced basis $V^H$ and truncate it as it is set in the input. The depth of the tree is set. Basis $V^H$ is split to $V^h$ using the refinement mechanism. For the chosen refined Basis $V^h$ the online stage is executed and relative error is computed as

$$\frac{1}{t}\sum_{k=1}^{t}\frac{\left\| u_{FOM}(\cdot,\tau^{k}) - u_{ROM}(\cdot,\tau^{k}) \right\|_{L^2}}{\left\| u_{FOM}(\cdot,\tau^{k}) \right\|_{L^2}} \tag{7.6}$$

where of difference between ROM and FOM is normalized using $L^2$ norm at the each time step $\tau$


Begin ROM
Insert:
- number of time steps $T^{MAX}$
Set number of iterations $It^{MAX}$
Compute Residual force vector
Compute Stiffness matrix
Apply boundary conditions
Compute reduced Stiffness matrix
Compute Reduced residual vector
Solve the system
Project displacement field
Compute residual $\boldsymbol{\varepsilon}$
Convergence check
$\boldsymbol{\varepsilon}$ < Tolerance
$It < It^{MAX}$
No
Yes
Project displacement field
Time step > $T^{MAX}$
No
Yes
Compute relative error
End ROM

***Flowchart 5.*** *Reduced Order Model*

# 8 APPENDIX B. MATLAB CODE

## 8.1 FULL ORDER MODEL

Corresponds to Algorithm 1

```
clear all
close all

addpath('make_R');
addpath('make_dRdu');
addpath('make_R/perzyna');


%% Input model, material, boundary conditions

[model,mat,bc] = input_values();


%% Initiate model

[ model, bc ] = initiate( model, mat, bc );
pd = [0 0]; %!!! initial load displacement curve

%% Time integrator

%initialize time integrator
[ time_step, u0, R_int, ut, ft, inVar_timeStep, inVar_iter ] =
initiate_timeAdvancing( model, mat );

% start counting time
tic;

for t=1:model.nT
    fprintf('Time step %i \n', t);

    % initialize Newton Raphson scheme
    du = zeros(model.sdof,1);

    % compute external part of residual vector
    R_ext = make_Rext( model, bc, t);

    for iter=1:model.niter

        % Residual vector (force vector)
        [R_int, inVar_iter] = make_Rint(model, mat, du, inVar_iter,
inVar_timeStep);
```

```
        % Jacobian matrix (stiffness matrix)
        dRdu = make_dRdu(model, mat, inVar_iter);

        R = R_int + R_ext;

        % apply boundary condition and initial value
        [dRdu, R] = apply_Boundary(dRdu, R, bc, iter, model, t);

        % solver
        ddu = - dRdu \ R;

        %update solution
        du = du + ddu;

        % convergence check
        if iter==1
            du1 = ddu;
        end

        res = norm(ddu)/norm(du1);
%         res = norm(R);
        fprintf('\t iteration %i , residual %i \n', iter, res);

        if res < model.tol
            break
        end

    end

    %update internal variables for current time step
    inVar_timeStep = update_inVar_timeStep(inVar_iter);

    u0 = u0+du;


    % collect sample
    ut(:,t) = u0;
    ft(:,t) = R_int;

    %% Post process, plot force vs displacement
    if t==1
        plot(0, 0, 'ob')
        hold on
    end
    pd = [pd ; u0(end), - R_int(end)];
    plot(pd(end, 1), pd(end,2), 'ob')
    drawnow

    %!!! stop if displacement reaches a certain point (any better way to
specify this ?)
    if u0(bc.dirichlet.dof.x(end)) > bc.dirichlet.dof.u(end) || -
R_int(bc.dirichlet.dof.x(end)) < 0;
        break
    end
end
```

```
xlabel('Displacement')
ylabel('Force')

timeStamp = toc;

fprintf('\nSimulation time: %i \n', timeStamp);

%% Post process
% save samples (snapshot data)
 save('/data/ut.mat', 'ut')
 save('/data/pd.mat', 'pd')
```

## 8.2 TREE CONSTRUCTION

The code corresponds to Flowchart 4

```
clear all
close all

%% INPUT (Snapshot matrix N_snapshots X N_DOF)
   load('/data/ut.mat');
%compute size of the basis
n_DOF=size(X,1);
n_samp=size(X,2);

XX=X;

%% NORMALIZATION (if needed)
%%Normalize observations (DOFs) to capture correlation%%%%%%

for i=1:size(XX,1) %through DOF

    if norm(XX(i,:),'fro') == 0 %if Zero cannot normalize

       Normal_XX(i,:)=XX(i,:);

    else                         %if nonzero - normalize

        for j=1:size(XX,2)

            Normal_XX(i,j)=XX(i,j)/norm(XX(i,:),'fro');

        end


    end
    if Normal_XX(i,1)<0 %flip origin if negative
        Normal_XX(i,:)=-Normal_XX(i,:);
    end

end

%END normalization%%

%% If no normalization was used
Normal_XX=XX;


%% CLUSTERING

%%_ Clustering by Matlab (Binary clusters Bottom-Up)

PwDist_XX=pdist(Normal_XX); %Compute PairwiseDistance (Euclidean) from
snapshot matrix
% squareform(PowDist_XX) %if Possible to see intercorrelaton between dof
i and j
LinkXX=linkage(PwDist_XX); %Grouping into binary clusters
SZLinkXX=size(LinkXX);

E={}; %Define contetnts of each cluster

%Assign to first NDOF clusters itself
for i=1:n_DOF
    E{i}=i;
    E_Err_dist{i}=0;
end

%Create the hierarchical clusters
for i=1:SZLinkXX(1) %Go vertically trough new clusters

    ResL=[]; %take left binary cluster
    ResR=[]; %take right binary cluster

    %For left side of linkage
    if LinkXX(i,1)>n_DOF %if more than NDOF than define what's inside
        ResL=E{LinkXX(i,1)};
    else ResL=LinkXX(i,1);
    end

    %For right side of linkage
    if LinkXX(i,2)>n_DOF %if more than NDOF than define what's inside
        ResR=E{LinkXX(i,2)};
    else ResR=LinkXX(i,2);
    end

    %Put right and left together
        E{n_DOF+i}=([ResL  ResR])  ;
        %Definition of each cluster (by DOFs)
        E_Err_dist{n_DOF+i}=LinkXX(i,3) ;
        %Store the distance (metric)
end
```

```
%Definition of each cluster (by other NDOF Clusters)
for i=1:(SZLinkXX(1))
    ClusN(((SZLinkXX(1)+n_DOF+1)-i),1:2)=LinkXX(((SZLinkXX(1)+1)-i),1:2);
end

EEE{1}=(SZLinkXX(1)+n_DOF); %First branch of the tree contains TOP
cluster number

EEE_sort{1}=(SZLinkXX(1)+n_DOF); %First branch of the tree contains top
cluster number (Sorted)

for i=1:(SZLinkXX(1)) %Go through clusters which are > than NDOF

    corrr=zeros(length(EEE{i}),n_DOF); %assign lenght of branch

    for j=1:(length(EEE{i})) %Create branches subsituing clusters with
entities

        ReadE=EEE{i}; % What is the cluster number?

        Val=strrep(ReadE(j),ReadE(j),E{ReadE(j)}); %Substitute cluster
number with its contetnt

            for mm=1:length(Val)
                %Fill sparse matrix for each tree level
                corrr(j,Val(mm))=Val(mm);
            end

    end
     TREE{i}=corrr; %Ready matrix for the branch (Entire tree is stored)

     cl=((SZLinkXX(1)+n_DOF+1)-i);

     expr=(ClusN(cl,:)); %Take out cluster numbers for the level

     EEE{i+1}=(strrep(EEE{i},cl,(expr))); %Branches created with clusters
names (globally)
     EEE_sort{i+1}= sort(strrep(EEE{i},cl,ClusN(cl,:))); %Branches
created with clusters names |Sorted|

    display(['Creation of branch E ' int2str(size(TREE,1))]);

end

%%%%Tree is alredy created in EEEL_1
save('data\Tree_E.mat','TREE','-v7.3')

%% PLOT DENDROGRAM
figure
[H]=dendrogram(LinkXX,'ColorThreshold','default'); %Plot the tree
Dendrogram
 xlabel('DOF indexes, clustered')
 ylabel('Distance between objects')
%
%
```

```
%% Compute projection error for each branch
%
% %%%%Select coarse basis%%%%%
%           tic
[V,SIG,~]=svd(Normal_XX);

n_basis=150; %set number of basis

for curr_bas=[5 10 25 50 100 150 200 250]

    display(['Basis ' int2str(curr_bas)]);

         V_H=-V(:,(1:curr_bas));        % take first column of SVD
approximation (Coarse Basis)
%
 V_h=[];
 V_h_norm=[];

for n_branch=[1 2 3 10 50 100 200 250] %choose number of branches
    display(['Branch ' int2str(n_branch)]);

E_S=TREE{n_branch}; %Selected DEPTH of tree

    %%%%Refine corse basis using the tree%%%%%

            V_h=[]; % Define sparse matrix for the refined basis


            for j=1:(size(V_H,2)) %through size of reduced basis

                for i=1:(size(E_S,1)) %through number of tree roots

                    E_I=nonzeros(E_S(i,:))';
                    V_h_1(E_I,i)= V_H(E_I,j); %create matrix of basis due
to tree of one root

                end

                   V_h=[V_h V_h_1]; %create matrix of basis due to tree
of all roots
                   V_h_1=0*[];

            end


%% RRQR See Flowchart 2 of Code Manual)

    V_h_RRQR=V_h;
    rrqr_tol = 1e-6;   %RRQR tolerance
    cols=rrqr(V_h,rrqr_tol); %linarly dependant columns indexes
    V_h_RRQR(:,cols)=[];  %take them out

X=Normal_XX;
%
```

```
%% Compute projecion Error:
%--Corresponds to Flowchart 3 of Code Manual--
            %after RRQR
            Err_pinv_REF_RRQR   (curr_bas,n_branch)  = norm((X-
V_h_RRQR*pinv(V_h_RRQR)*X),'fro')/norm(X,'fro');
            %normalization+RRQR
            Err_pinv_REF_normXX_RRQR   (curr_bas,n_branch)  =
norm((Normal_XX-
V_h_RRQR*pinv(V_h_RRQR)*Normal_XX),'fro')/norm(Normal_XX,'fro');

 end
%
end%
```

## 8.3 Refinement mechanism

Function split basis corresponds to Flowchart 1

```
function [va]=splitbasis(V_H,n_branch)


    %load Tree structure
    load(/Tree_E.mat','TREE')


    E_S=EEEL_1{n_branch}; %Selected DEPTH of tree

    %%%%Refine coarse basis using the tree%%%%%%

    V_h=[]; % Define sparse matrix for the refined basis

        for j=1:((size(V_H,2))) %through size of reduced basis

            for i=1:((size(E_S,1))) %through number of tree roots

                    E_I=nonzeros(E_S(i,:))';

%                   create matrix of basis due to tree of one root
                    V_h_1(E_I,i) = V_H(E_I,j);

                end

                   %create matrix of basis due to tree of all roots
                   V_h=[V_h V_h_1];

                   V_h_1=0*[];

        end
```

```
%% RRQR (See Flowchart 2 of Code Manual)

    va=V_h;

    %input RRQR tolerance
    rrqr_tol = 1e-10;

    %find indexes of linarly dependant columns
    cols=rrqr(V_h,rrqr_tol);

    %take linarly dependant columns out
    va(:,cols)=[];  %take them out

end
```

## 8.4 Rank Revealing QR factorization

RRQR algorithm is explained by Flowchart 2

```
function [jout] = rrqr(A,tol)

rho = norm(A(:,1),2); %Norm of 1st col w.r.t. to row
q1  = A(:,1) / rho; %normalized 1st column

Q       = [q1];
jout    = []; %index of columns to be taken out

for l=2:size(A,2)
    r = Q' * A(:,l);
    q = A(:,l) - Q*r;

    rho = norm(q,2); %rho = norm(A(:,l) - Q*(Q' * A(:,l);),2)

    if rho/norm(r,2) < tol
        jout = [jout, l]; %store index
    else
        Q = [Q, q/rho]; %append column
    end

end

end
```

## 8.5 Reduced Order Model

Corresponds to Flowchart 5

```
clear all
close all

addpath('make_R');
addpath('make_dRdu');
addpath('make_R/perzyna');

%% Input model, material, boundary conditions

[model,mat,bc] = input_values();

%% Initiate model

[ model, bc ] = initiate( model, mat, bc );

% load values of FOM
  load ('/data/ut.mat', 'ut')
  ut_FOM = ut;
  XX = ut;

% SVD of the snapshot matrix
 [va,SIG,~] = svd(XX);

% store left orthogonal vector V_H
 va_ini = va;

for  n_bas = 1: 5 %SET number of basis

for n_branch = 1: 80 %SET depth of tree
pd = [0 0]; % initial load displacement curve

va=va_ini;

fprintf('Chosen basis: %i \n', n_bas);
fprintf('Chosen branch: %i \n', n_branch );

%% time integrator

%initialize time integrator
[ time_step, u0, R_int, ut, ft, inVar_timeStep, inVar_iter ] =
initiate_timeAdvancing( model, mat );


% start counting time
tic;

% truncate basis V_H
```

```
            va_1 = va(:,(1:n_bas));

% split V_H to obtain V_h (see Refinement Mechanism)

            Va = splitbasis(va_1,n_branch);

% start MOR
for t=1: model.nT
    fprintf('Time step %i \n', t);

    % initialize newton raphson scheme
    du = zeros(model.sdof,1);

    % compute external part of residual vector
    R_ext = make_Rext( model, bc, t);


    for iter=1:model.niter

        % Residual vector (force vector)
        [R_int, inVar_iter] = make_Rint(model, mat, du, inVar_iter,
inVar_timeStep);

        % Jacobian matrix (stiffness matrix)
        dRdu = make_dRdu(model, mat, inVar_iter);

        R = R_int + R_ext;

        % apply boundary condition and initial value
        [dRdu, R] = apply_Boundary(dRdu, R, bc, iter, model, t);

        % compute the reduced stiffness matrix and the reduced force
vector
            % Reduced stiffness
            dRdu_r=va'*dRdu*va;

            % Reduced residual
            R_r=va'*R;

            % Solver
            ddu_r = - dRdu_r \ R_r;

            % Project back
            ddu = va * ddu_r;

        % update solution
        du = du + ddu;

        % convergence check
        if iter==1
            du1 = ddu;
        end
        % check convergence
        res = norm(va'*ddu)/norm(va'*du1);

        fprintf('\t iteration %i , residual %i \n', iter, res);

        if res < model.tol
            break
        end

    end

    %update internal variables for current time step
    inVar_timeStep = update_inVar_timeStep(inVar_iter);

    u0 = u0+du;

    %compute relative error
    err_ut(t) = (norm((ut_FOM(:,t)-u0),'fro')/ norm(ut_FOM(:,t),'fro'));

end

 %store size of the basis
 store_sz_basis(n_branch,n_bas) = size(va,2);

 %store relative error
 store_pd_displ_Vh(n_branch,n_bas) = (1/size(ut_FOM,1))*sum(err_ut);

end

end

timeStamp = toc;
```

## REFERENCES

[1] P. Krysl, S. Lall and J. E. Marsden, "Dimensional model reduction in non-linear finite element dynamics of solids and structures," *Int J Numer Methods Eng,* vol. 51, pp. 479-504, 2001.

[2] D. Amsallem and C. Farhat, "Interpolation Method for Adapting Reduced-Order Models and Application to Aeroelasticity," *Aiaa J.,* vol. 46, pp. 1803-1813, 07/01; 2017/03, 2008.

[3] A. Doostan, R. G. Ghanem and J. Red-Horse, "Stochastic model reduction for chaos representations," *Comput. Methods Appl. Mech. Eng.,* vol. 196, pp. 3951-3966, 8/1, 2007.

[4] K. Carlberg, C. Farhat, J. Cortial and D. Amsallem, "The GNAT method for nonlinear model reduction: Effective implementation and application to computational fluid dynamics and turbulent flows," *Journal of Computational Physics,* vol. 242, pp. 623-647, 6/1, 2013.

[5] C. W. Rowley, "Model Reduction for fluid, using balanced Proper Orthogonal Decomposition," *Int. J. Bifurcation Chaos,* vol. 15, pp. 997-1013, 03/01; 2017/03, 2005.

[6] P. Perzyna, "Fundamental Problems in Viscoplasticity," *Adv. Appl. Mech.,* vol. 9, pp. 243-377, 1966.

[7] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics,* 1967, pp. 281-297.

[8] J. M. Burgers, "A Mathematical Model Illustrating the Theory of Turbulence," *Adv. Appl. Mech.,* vol. 1, pp. 171-199, 1948.

[9] Y. Wang, I. M. Navon, X. Wang and Y. Cheng, "2D Burgers equation with large Reynolds number using POD/DEIM and calibration," *Int. J. Numer. Methods Fluids,* vol. 82, pp. 909-931, 2016.

[10] D. Citrini and G. Noseda, *Idraulica.* CEA, 1987.

[11] E. Hopf, "The partial differential equation ut + uux = μxx," *Communications on Pure and Applied Mathematics,* vol. 3, pp. 201-230, 1950.

[12] T. Korenius, J. Laurikkala and M. Juhola, "On principal component analysis, cosine and Euclidean measures in information retrieval," *Inf. Sci.,* vol. 177, pp. 4893-4905, 11/15, 2007.

[13] G. Qian, S. Sural, Y. Gu and S. Pramanik, "Similarity between euclidean and cosine angle distance for nearest neighbor queries," in *Proceedings of the 2004 ACM Symposium on Applied Computing,* Nicosia, Cyprus, 2004, pp. 1232-1237.

[14] J. Wu, *Advances in K-Means Clustering: A Data Mining Thinking.* Springer Publishing Company, Incorporated, 2012.

[15] K. Carlberg, "Adaptive h-refinement for reduced-order models," *Int J Numer Methods Eng,* vol. 102, pp. 1192-1210, 2015.

[16] Y. P. Hong and C.-T. Pan, "Rank-Revealing QR Factorizations and the Singular Value Decomposition," *Mathematics of Computation,* vol. 58, pp. 213-232, 1992.

[17] T. F. Chan. Rank revealing QR factorizations. *Linear Algebra and its Applications 88*pp. 67-82. 1987.