Imperial College London

Department of Earth Science and Engineering

MSc in Applied Computational Science and Engineering

Independent Research Project
Final Report

Inverse Analysis of Planar Bodies

by

Iñigo Basterretxea Jacob

ib616@imperial.ac.uk
GitHub login: acse-ib616

Supervisors:
Dr. Adam J. Sadowski

August 2021

GitHub repository: `https://github.com/acse-ib616/SHELLS.git`

Abstract

The reconstruction of applied load configurations in structures is a specific type of inverse problem with structure-monitoring and forensic applications in the aeronautical, automobile and civil engineering industries. It has been a fertile topic of discussion in structural engineering in the last two decades, with new approaches such as the inverse finite element method having contributed to significant advances. Most efforts compute the applied nodal force vector, but this does not provide a complete understanding of the boundary loads as multiple load cases may yield the same forces at the nodes. Hence, a method is devised to optimise for a set of element edge and surface loads. The approach involves providing a nodal forces vector and assembling the stiffness matrix to compute the nodal forces vector. Then a Jacobian that combines the equivalent nodal forces and a set of element loads is formulated and used to perform a least-squares minimisation with the Gauss-Newton method. Where the problem is ill-conditioned, the Levenberg–Marquardt modification is proposed. C++ and MATLAB are combined to use each language's strengths taking into account run-time speed as well as practical code development considerations. This method, which originally was meant to consider shell elements, has been validated for the less complex truss, constant strain triangle and linear strain triangle elements by providing a set of input edge loads and comparing it with the output of the Gauss-Newton algorithm. The main advantage of this approach is that it can be applied to different elements, as the Jacobian coefficients are dependent on the element loads and are easily calculated. Subsequent research opportunities suggested include the combination of the inverse finite element method with the technique proposed in the study, as well as the consideration of non-uniform element forces.

Contents

1 Introduction

Structural engineers are usually concerned with designing structures that can withstand certain loading configurations. They often do so by carrying out a structural analysis with the help of the Finite Element Method (FEM) to assess the structural response of their designs and ensure it meets design requirements. However, there are cases where one may be interested in the inverse problem: determining the forces that caused a specific structural response. An example of this is forensic engineering, where engineers must establish what caused a failure or a performance issue. Another example is when real-time information about a structure's loading and its performance is monitored, as is the case in the aeronautical and automobile industries.

A particularly challenging type of structural element is the thin shell, for which in contrast to other structural elements such as beams, walls or slabs, it is generally difficult to predict their behaviour under loading. Thin-walled shell elements are curved, with a thickness that is small compared to its radii and in which deformations are not large compared to their thickness (Sadowski 2019). Performing an inverse analysis on shells is a computationally expensive task that requires optimised tools, as several iterations contemplating thousands of degrees of freedom (DOFs) might be needed to reach a solution.

This report presents the implementation of a Finite Element (FE) solver for planar bodies, which are one of the building blocks required to develop shell formulations, with the other being the plate element. The formulation of increasingly more complex 2D elements will be studied to understand their limitations. Additionally, an approach for performing a load reconstruction on 2D bodies that can be generally used for other types of elements. This piece of work should be understood as a preliminary step in the development of an inverse analysis 3D shell finite element model.

2 Literature Review

2.1 Constant Strain Triangle (CST)

2.1.1 Formulation

The simplest formulation of an element for 2D stress analysis is the CST element (Macorini 2019), which includes 6 displacement DOFs, with a vertical and horizontal displacement at each of its 3 nodes. Its name derives from the displacement field being linear across the element, which results in a constant strain field.

$$u(x, y) = a_1 + a_2x + a_3y \tag{1}$$

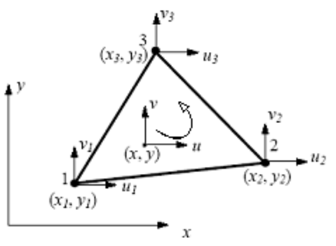

Figure 1: CST Element (GeoStru n.d.).

In order to derive the element formulation, a local coordinate system may be used.The global (x, y) and local (r, s) coordinates may be related as follows:

$$x = \sum_{i=1}^{3} x_iN_i = N_1x_1 + N_2x_2 + N_3x_3 \tag{2}$$

where the linear shape functions are defined as

$$N_1 = 1 - r - s,\ N_2 = r,\ N_3 = s \tag{3}$$

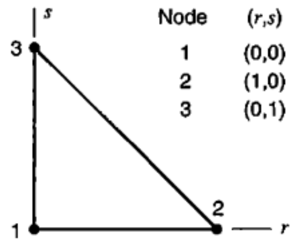

Figure 2: Local coordinate system (Cook et al. 2007).

Equally, the element displacement fields may be related to the nodal displacements with shape functions:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \{u\} = [N]\{U\} = \begin{bmatrix} N_1 & 0 & N_2 & 0 & N_3 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \end{Bmatrix} \tag{4}$$

The strains, assuming a geometrically linear analysis, are determined using the following kinematic relations:

$$\begin{Bmatrix} \epsilon_x \\ \epsilon_y \\ \gamma_{xy} \end{Bmatrix} = \{\epsilon\} = [\partial]\{u\} = \begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} \tag{5}$$

which, substituting Equation 4, becomes

$$\{\epsilon\} = [B]\{U\},\ \text{where } [B] = [\partial][N] \tag{6}$$

However, since the shape functions are function of the local coordinates (r, s), a coordinate transformation will be necessary. This is achieved using the Jacobian matrix, defined as:

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial r} & \frac{\partial y}{\partial r} \\ \frac{\partial x}{\partial s} & \frac{\partial y}{\partial s} \end{bmatrix} = \begin{bmatrix} \frac{\partial N_1}{\partial r} & \frac{\partial N_2}{\partial r} & \frac{\partial N_3}{\partial r} \\ \frac{\partial N_1}{\partial s} & \frac{\partial N_2}{\partial s} & \frac{\partial N_3}{\partial s} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \end{bmatrix} = \begin{bmatrix} J_{11} & J_{12} \\ J_{21} & J_{22} \end{bmatrix} \tag{7}$$

such that $[\partial]$ may be expressed as

$$\begin{bmatrix} \frac{\partial}{\partial x} & 0 \\ 0 & \frac{\partial}{\partial y} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} \end{bmatrix} = \frac{1}{|J|} \begin{bmatrix} J_{22}\frac{\partial}{\partial r} - J_{12}\frac{\partial}{\partial s} & 0 \\ 0 & -J_{21}\frac{\partial}{\partial r} + J_{11}\frac{\partial}{\partial s} \\ -J_{21}\frac{\partial}{\partial r} + J_{11}\frac{\partial}{\partial s} & J_{22}\frac{\partial}{\partial r} - J_{12}\frac{\partial}{\partial s} \end{bmatrix} \tag{8}$$

where $|J|$ is the determinant of the Jacobian. Hence, $[B]$ may be derived using Equation 8 and Equation 6 expressed as

$$[B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \frac{\partial N_3}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & 0 & \frac{\partial N_3}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} \end{bmatrix} \tag{9}$$

The stresses are then related to the strains via the material constitutive relations:

$$\{\sigma\} = [D]\{\varepsilon\} \tag{10}$$

where $[D]$ is the constitutive matrix, which for the CST element is defined by the plane-stress condition:

$$[D] = \frac{E}{1-\nu^2}\begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \frac{1-\nu}{2} \end{bmatrix} \tag{11}$$

Using the principle of virtual work, a relation between the element nodal displacements and the nodal resistance forces may be obtained (CITE):

$$\{f\} = [k]\{U\} - \{f_p\} \tag{12}$$

where the element stiffness matrix $[k]$ is a 6×6 matrix defined as

$$[k] = \int [B]^T[D][B]dV \tag{13}$$

which, assuming constant thickness and $[D]$ throughout the element, becomes

$$[k] = \frac{t}{2}\int_0^1\int_0^1 [B]^T[D][B]drds \tag{14}$$

Now, for the CST element matrix $[B]$ is not dependent on local coordinates (r, s) as all its coefficients are constants. Hence, the element stiffness matrix is simply calculated as

$$[k] = \frac{t}{2}[B]^T[D][B] \tag{15}$$

2.1.2 Limitations

The CST element presents excessive stiffness in bending due to the presence of a spurious shear strain (Streamliner n.d.). This additional shear stress helps the element reach equilibrium at smaller displacements and is introduced as the linear element sides are not able to bend and mimic the curvature caused by bending action (Sun 2006). In FEA this is known as *shear locking*. Due to this, FE models using the CST element need a high mesh resolution to converge.

2.2 Linear Strain Triangle (LST)

Moving on to the immediately more complex element, the LST includes 12 translational displacement DOFs distributed among its 6 nodes. Similarly to the CST its name derives from the displacement field being quadratic across the element, resulting in a linear strain field.

$$u(x, y) = a_1 + a_2x + a_3y + a_4x^2 + a_5y^2 + a_6xy \tag{16}$$

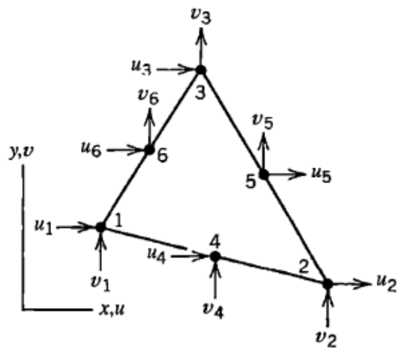

Figure 3: LST Element (Cook et al. 2007).

Once again, a local coordinate system (r, s) should be used to derive the element stiffness matrix, with the global coordinate system being related to the local one as described before

$$x = \sum_{i=1}^{6} x_i N_i = N_1 x_1 + N_2 x_2 + N_3 x_3 + N_4 x_4 + N_5 x_5 + N_6 x_6 \tag{17}$$

where the shape functions are now quadratic functions

$$N_1 = 1 - r - s,\ N_2 = 2r^2 - r,\ N_3 = 2s^2 - s$$
$$N_4 = -4r^2 - 4rs + 4r,\ N_5 = 4rs,\ N_6 = -4rs - 4s^2 + 4s$$

All in all, the derivation of the element stiffness matrix follows the same logic, where the kinematic and constitutive relations are used in conjunction with the principle of virtual work to obtain a relation between nodal resistance forces and nodal displacements, where the stiffness matrix is obtained by integrating over the element domain. However, the coefficients in the matrix $[B]$ as well as in the Jacobian are now dependent on the local coordinates (r, s). This makes the resulting double integral very cumbersome. In fact, such is the level of complexity of integrating the 12×12 matrix resulting from the $[B]^T[D][B]$ operation that even a powerful symbolic computing environment such as Maple cannot handle it. Thus, numerical integration will be needed to obtain the element stiffness matrix.

2.3 Triangular Element with Drilling DOF (TR)

A slightly improved version of the triangular element is the triangular element with a drilling DOF (TR), that is, a rotation about the direction normal to the plane of the element. It is an attractive element for engineers as it reduces the number of degrees of freedom per element by removing side nodes, whilst still providing a satisfactory level of performance. Further, it is of particular interest in shell analysis given that the simplest form of shell elements, which is the one generated by a plane element (e.g. the CST) and a plate bending element with 5 DOFS at each node (three translational and two rotational), would not normally include this sixth DOF. Including this additional rotational DOF allows to account for the contribution to in-plane performance arising from in-plane rotational stiffness and also alleviates problems such as singular matrix systems (Jin 1994) or the difficult assembly of shell elements. (Cook 1993).

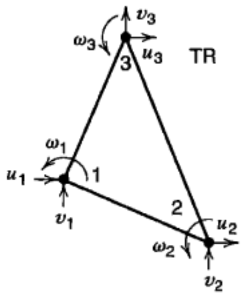

Figure 4: TR Element (Cook et al. 2007).

The TR element can be obtained by degenerating the LST element by transforming the translational DOFs at the side nodes into drilling DOFs at the vertices of the triangle. Assume a deviation at the side of a triangular plane element, normal to the side and quadratic along the side-length produced by rotational DOFs ω_i and ω_i at their respective nodes i and j:

$$\delta = \frac{s(L - s)}{2L}(\omega_j - \omega_j) \tag{18}$$

such that the midspan deviation is

$$\delta_m = \frac{L}{8}(\omega_j - \omega_j) \tag{19}$$

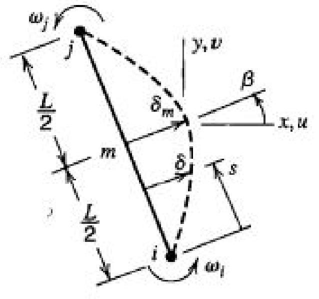

Figure 5: Side-displacement due to drilling DOF (Cook et al. 2007).

From Figure 2.3, we observe that δ may be decomposed in terms of sine and cosine of angle β into midspan displacement components

$$\delta_m \cos\beta = u_m \text{ and } \delta_m \sin\beta = v_m \tag{20}$$

the former of which in turn can be expressed as

$$L\cos\beta = y_j - y_i \text{ and } L\sin\beta = x_i - x_j \tag{21}$$

By considering the midspan displacement components as the average between node translational DOFs, we obtain the following relation

$$\begin{Bmatrix} u_m \\ v_m \end{Bmatrix} = \frac{1}{2}\begin{Bmatrix} u_i \\ v_i \end{Bmatrix} + \frac{1}{2}\begin{Bmatrix} u_i \\ v_j \end{Bmatrix} + \frac{1}{8}(\omega_j - \omega_i)\begin{Bmatrix} y_j - y_i \\ x_j - x_i \end{Bmatrix} \tag{22}$$

Thus, we have established a relationship between the DOFs in the LST element and those in the TR element with the transformation matrix $[T]$:

$$\begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \\ u_5 \\ v_5 \\ u_6 \\ v_6 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1/2 & 0 & -y_{21}/8 & 1/2 & 0 & y_{21}/8 & 0 & 0 & 0 \\ 0 & 1/2 & -x_{21}/8 & 0 & 1/2 & x_{21}/8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 0 & -y_{32}/8 & 1/2 & 0 & y_{32}/8 \\ 0 & 0 & 0 & 0 & 1/2 & -x_{32}/8 & 0 & 1/2 & x_{32}/8 \\ 1/2 & 0 & y_{13}/8 & 0 & 0 & 0 & 1/2 & 0 & -y_{13}/8 \\ 0 & 1/2 & x_{13}/8 & 0 & 0 & 0 & 0 & 1/2 & -x_{13}/8 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ \omega_1 \\ u_2 \\ v_2 \\ \omega_2 \\ u_3 \\ v_3 \\ \omega_3 \end{Bmatrix} \tag{23}$$

where $x_{ij} = x_i - x_j$ and $y_{ij} = y_i - y_j$ Therefore, we can simply obtain the TR element stiffness matrix $[k]$ by applying the transformation to the LST element stiffness matrix $[k']$

$$[k] = [T]^T[k'][T] \tag{24}$$

2.4 Load Reconstruction

Load reconstruction is a type of inverse problem where a structure's response to loading is known in either the form of strains or displacements, and the engineer would like to assess the applied forces that caused it. For simple systems, analytical solutions are easily derived from the theory, but for more complex ones, the finite element method must be inevitably involved in the solution.

Work on this subject is relatively recent, with most efforts revolving around the use of the inverse finite element method (iFEM) developed by Tessler (2003) and focused on determining the applied nodal forces, rather than the actual distribution of the applied loading. Literature on the latter is considerably more limited and is highlighted by the work carried out by Sanayei & Saletnik (1996), as well as Chock & Kapania (2003) and Li & Kapania (2007) who studied different load distribution functions. However, the only elements studied have been truss and beam elements, with no known attempts with CST, LST, plate or shell elements.

3 Problem Description & Objectives

This study aims to provide a tool that performs load reconstruction for a certain type of structures. Initially, thin-walled cylindrical shells were chosen. However, as the reader will realise, formulating even the simplest three-dimensional shell finite element is an onerous endeavour. Thus, it was decided that the scope of the study would be reduced to encompass two-dimensional bodies, which are one of the elements from which some shell elements are constructed. Nevertheless, the method developed should be be reproducible for other types of structures and finite elements, as this project focuses not only on delivering the tool itself but also on presenting a general approach to load reconstruction using the FEM. The load reconstruction must not be limited to obtaining the applied nodal loads, but actually encompass finding the element-wise load distributions whose assembly results in those forces at the nodes.

Finding the original load configuration is a type of inverse problem, and therefore an optimisation problem. As such, an objective function shall be defined, whose purpose is to evaluate the difference between the given structural response (e.g. the global nodal displacement vector) and the one output by the FEA tool developed.

FE software packages such as ABAQUS are very powerful and user-friendly, but unfortunately they cannot efficiently generate multiple FE models in quick succession whilst interacting with other pieces of software that would enable optimisation routines to be utilised. They will be helpful for validating the finite element models developed from scratch.

Inverse problems often require a large number of iterations and the finite element models might demand fine meshes to be accurate, so the software package developed must be highly optimised to reduce the computational cost and be useful. This is the type of task where a compiled language like C++ excels. The C++ FE package will then link with MATLAB via MEX files, as the latter will be used for handling inputs, post-processing, and whenever code development is accelerated without an efficiency cost.

4 Methodology

4.1 Assembling the stiffness matrix K

Regardless of the type of structure we are studying and the elements that constitute it, any problem can be described by a system of equations of the following form:

$$\{F\} = [K]\{U\} \tag{25}$$

where K is the global stiffness matrix, F is the global nodal forces vector and U is the nodal displacement vector. K is dependent on material properties such as the Young's Modulus E and the Poisson coefficient ν, geometric properties such as the thickness t, as well as the type of mesh elements used to analyse the structure.

The code only handles input in the form of rectangular structures, as they can be easily identified and assembled into the mesh. These can be built as a truss, which can be divided into bays and storeys; or as rectangular 2D bodies, which can be divided into triangular elements. The convention adopted is that nodes are numbered in column major order from bottom to top, starting at the leftmost edge of the body, and towards the right (see Figure 4.1). Each node has a set of x and y coordinates associated to it, as well a set of two DOF ID numbers. As for the elements, in the case of the CST these are numbered by taking the rectangles formed by two complementary triangular elements in row major order, from bottom to top, and from left to right (see Figure 6).

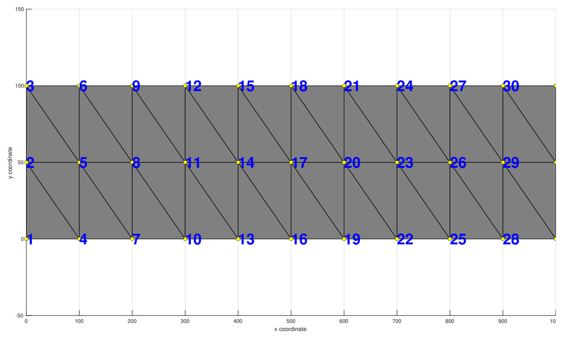

Figure 6: Node numbering.

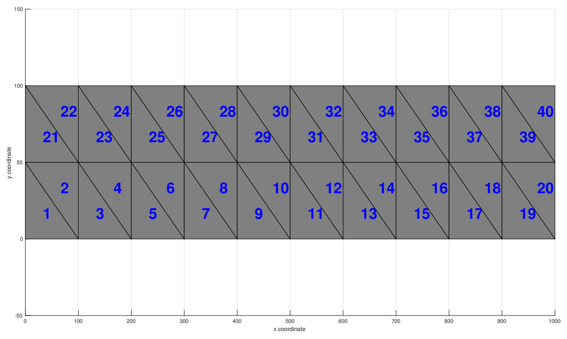

Figure 7: Element numbering for structures built from CST elements.

The global stiffness matrix K is built from the addition of local element stiffness matrices K_e. The latter have been coded with the help of the *Maple CodeGenerator*, which elegantly translates the symbolic mathematical expressions for K_e described in Section 2 into C++ code. The contribution of each element to the global stiffness matrix K is found by looping through each element, identifying the nodes, and hence their corresponding coordinates and DOFs. DOF IDs indicate the position of the contribution of each local element DOF within the global K.

Now, stiffness matrices have a high sparsity, which is a beneficial feature when operating with large meshes. To take advantage of this, the global stiffness matrix K will have to be assembled sparsely. In this connection, the compressed sparse row (CSR) matrix is a popular format for storing sparse matrices. The format is composed of three vectors: the first contains the non-zero values, the second includes the column indices of these values, and the third contains the position of the first non-zero value of each row in the first vector. Calculating each of the coefficients of K in a sparse manner requires knowing where the non-zero values are located. Therefore, initially the indices of the non-zero values are obtained by looping though all the elements, without calculating the coefficients, so that the "column indices" and the "row positions" vectors are computed. Once the indices of the non-zero values are known, the coefficients are calculated by looping through all the elements, getting the local element stiffness matrices and adding the contributions of each local DOF that matches a DOF ID in the global K.

A stiffness matrix assembler was developed for both the CST, the LST and the TR elements. These were validated by carrying out a mesh convergence study with the simple case of a rectangular plate acting as a simply-supported beam under an imposed uniformly-distributed load (UDL) and measuring the vertical displacement at midspan. The results were compared to Euler-Bernoulli beam theory and the output from ABAQUS FEA (see Figure 4.1). The mesh convergence study showed that the finite element formulations for both the CST and the LST elements - the TR element is not included in the ABAQUS package - were correct, as they showed very good agreement with the ABAQUS results. Interestingly, the TR element seemed to be stiffer than the CST for a lower mesh resolution.

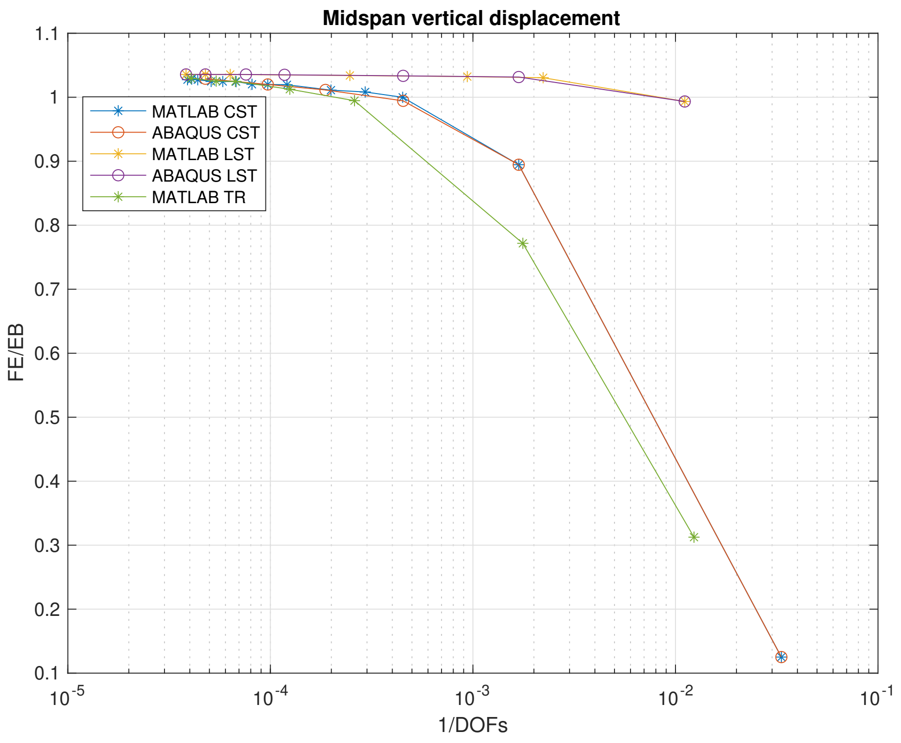

Figure 8: Mesh convergence. The vertical axis is the ratio of vertical displacement at midspan from FEA over that of Euler-Bernoulli theory.

4.2 Assembling the global nodal force vector F

The global nodal force vector represents the combined action of the equivalent nodal loads F_{eq} and the applied nodal forces F_a.

$$\{F\} = \{F_a\} + \{F_{eq}\} \tag{26}$$

The former comprises loads applied directly at the loads, including reaction forces; whilst the latter includes forces that are not directly applied at the nodes, such as element edge forces or body forces. Non-nodal forces must be resolved into nodal forces in order to include their effects. This is easily done in the case of uniform forces. For instance for a CST element, a uniformly distributed load may be resolved into two point loads at the two end nodes as

$$f_p = \frac{wh}{2} \tag{27}$$

where w is the value of the UDL in N/m and h is the length of the edge. As for uniform body forces, such as self-weight, the equivalent nodal loads will be distributed amongst all three nodes of the element as

$$f_p = \frac{\gamma A t}{3} \tag{28}$$

where γ is the value of the body force in N/m^3, A is the area of the element and t its thickness.

Similarly to the case of the global stiffness matrix K, the global nodal force vector F is assembled by looping through all the elements. Its assembly is considerably easier due to being a one-dimensional matrix and thus not being sparse. For this reason, its assembly did not require the speed provided by C++ coded and was simply done in MATLAB.

4.3 Solving the Equations

Once the nodal forces vector F and the stiffness matrix K are assembled, forming a linear system of equations, its solution should yield the nodal displacement vector U. The right hand side should include the applied and equivalent forces, as well as the reaction forces F_R, yet these last ones are unknown, so the system cannot be solved directly. The system must be partitioned such that the free DOFs and the restrained DOFs give a set of submatrices and subvectors, which allow to represent the system as follows:

$$\left[\begin{array}{cc}[K_{FF}] & [K_{FR}] \\ \hline [K_{RF}] & [K_{RR}]\end{array}\right] \begin{Bmatrix}\{U_F\} \\ \{U_R\}\end{Bmatrix} = \begin{Bmatrix}\{F_F\} \\ \{F_R\}\end{Bmatrix} \tag{29}$$

This system can therefore be thought of as two separate matrix equations. With the first, the nodal displacements at the free nodes U_F can be computed:

$$[K_{FF}]\{U_F\} + [K_{FR}]\{U_R\} = \{F_F\} \tag{30}$$

Whilst the solution to the second equation results in the reaction forces F_R:

$$[K_{RF}]\{U_F\} + [K_{RR}]\{U_R\} = \{F_R\} \tag{31}$$

Note that the matrix inversion only needs to take place with the K_{FF} submatrix in the first equation, as the second one directly yields the unknowns.

MATLAB's linear solvers are efficient and its matrix environment makes matrix operations and indexing straight-forward. Given that the potential improvement in efficiency resulting from the onerous task of solving the system in C++ is minimal, as most of the code optimisation can be achieved by storing K sparsely; extracting the submatrices mentioned above, as well as arranging and solving the matrix equations is carried out with MATLAB.

4.4 Optimisation: Gauss-Newton & Levenberg–Marquardt Modification

When trying to obtain the solution to the loading configuration that caused a certain structural response, the obvious way to proceed would be to assemble a new nodal forces vector using a different set of loads at every iteration, solve the linear system of equations to obtain an output nodal displacement vector and compare the displacement vector obtained with the given one. However, this method involves inverting the stiffness matrix K for every iteration, which is computationally expensive. Instead, a more cost-effective approach would be to optimise for the loading configuration comparing the global nodal forces vector.

The global nodal forces vector includes the reaction forces at the restrained nodes. Computing those requires inverting the stiffness matrix, which would defeat the purpose of this approach. Therefore, the optimisation will ignore the restrained DOFs to avoid this issue. Thus, the fitness function to be minimised will be:

$$R(\boldsymbol{q}) = \sum_{i=1}^{m} [Y_{free,i} - F_{free,i}(\boldsymbol{q})]^2 \tag{32}$$

where Y_{free} is the vector of equivalent nodal forces calculated from the known displacement vector U after multiplying by the stiffness matrix K, and $F_{free}(\boldsymbol{q})$ is the assembled vector of equivalent nodal forces dependent on the input set of loads $\boldsymbol{q}$.

One of the most popular methods for solving multivariate non-linear least squares problems is the Gauss-Newton algorithm. It is an iterative method where an initial guess for the set of parameters $\boldsymbol{q}$ must be provided and in successive iterations, the set of parameters $\boldsymbol{q}$ is replaced for a new estimate $\boldsymbol{q} + \boldsymbol{\Delta}$. In this case, the set of parameters we are optimising for is the set of loads $\boldsymbol{q}$. F_{free} may be linearly approximated with a Taylor expansion and expressed in vector form as (Chong & Zak 2004)

$$\boldsymbol{F_{free}}(\boldsymbol{q} + \boldsymbol{\Delta}) \approx \boldsymbol{F_{free}}(\boldsymbol{q}) + \frac{\partial \boldsymbol{F_{free}}}{\partial \boldsymbol{q}} \boldsymbol{\Delta} \tag{33}$$

where the the gradient of $\boldsymbol{F_{free}}$ with respect to q can be represented with the Jacobian $\boldsymbol{J}$, such that the expression is rewritten as

$$\boldsymbol{F_{free}}(\boldsymbol{q} + \boldsymbol{\Delta}) \approx \boldsymbol{F_{free}}(\boldsymbol{q}) + \boldsymbol{J}\boldsymbol{\Delta} \tag{34}$$

Using the linearisation of $\boldsymbol{F_{free}}(\boldsymbol{q})$ in Equation 33, the sum of the square of the residuals (Equation 32) may be expressed as

$$\boldsymbol{R}(\boldsymbol{q}) \approx [\boldsymbol{Y_{free}} - \boldsymbol{F_{free}}(\boldsymbol{q}) - \boldsymbol{J}\boldsymbol{\Delta}]^2 \tag{35}$$

This expression has a minimum when its gradient with respect to $\boldsymbol{q}$ is zero. Hence, after expanding it, differentiating it with respect to $\boldsymbol{q}$ and setting it to zero, the result is the following equation:

$$(\boldsymbol{J}^T\boldsymbol{J})\boldsymbol{\Delta} = \boldsymbol{J}^T[\boldsymbol{Y_{free}} - \boldsymbol{F_{free}}(\boldsymbol{q})] \tag{36}$$

After solving for $\boldsymbol{\Delta}$, the next estimate for $\boldsymbol{q}$ can be calculated:

$$\boldsymbol{q}_{i+1} = \boldsymbol{q}_i + \boldsymbol{\Delta} \tag{37}$$

If the $\boldsymbol{J}^T\boldsymbol{J}$ product is ill-conditioned and therefore not invertible, an improved version of the Gauss-Newton algorithm may be used to allow for the inversion, the Levenberg–Marquardt modification. This involves the addition of a diagonal matrix, such that the expression $\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I}$ becomes positive definite and can thus be inverted. The algorithm works best when the smallest possible value for λ is chosen such that the aforementioned expression becomes positive definite.

$$(\boldsymbol{J}^T\boldsymbol{J} + \lambda\boldsymbol{I})\boldsymbol{\Delta} = \boldsymbol{J}^T[\boldsymbol{Y_{free}} - \boldsymbol{F_{free}}(\boldsymbol{q})] \tag{38}$$

This modification was not necessary in the case of CST elements (i.e. $\lambda = 0$), whereas it had to be used for truss structures.

4.5 Assembling the Jacobian

A key component of optimisation algorithms for solving multivariate minimisation problems is the Jacobian matrix, which is defined as:

$$J = \begin{bmatrix} \frac{\partial F^1}{\partial x_1} & \cdots & \frac{\partial F^1}{\partial x_n} \\ . & . & . \\ . & . & . \\ . & . & . \\ \frac{\partial F^m}{\partial x_1} & \cdots & \frac{\partial F^m}{\partial x_n} \end{bmatrix} \tag{39}$$

In the case being analysed, the input variables are represented by the set of element edge loads, whereas the function being minimised is Equation 32. The latter is only dependent on the element loads. This is a key feature, which will allow the approach for finding the original loading configuration to be used for different types of finite elements, including shell elements.

To validate the approach taken the example described in Section 4.1 will once again be used. In such instance the coefficients of the Jacobian will be expressed as:

$$\frac{\partial F^1}{\partial x_1} = \frac{dx}{2} \tag{40}$$

Where dx is the length of the top edge of each element where the UDL is applied.

With the aim of speeding up the algorithm, the Jacobian matrix is assembled sparsely in C++ in a similar fashion as the global stiffness matrix K, firstly identifying the location of non-zero values and later calculating the coefficients.

4.6 MEX Functions: MATLAB to C++ & Back

Certain tasks such as assembling the sparse global stiffness matrix K and the Jacobian for the optimisation routine are quickly done in C++, where for-loops and simple arithmetic are performed considerably faster than in MATLAB. However, MATLAB's matrix environment is well suited for performing linear algebra operations, whilst its plotting libraries are more user-friendly than those of C++. MEX functions are used to extract the best features from both languages and obtain the best-performing piece of software.

MEX functions have arguments passed as a list of objects. These have are then unpacked and transformed into objects easier to work with such as vectors. In this process, it is important the object types are compatible between the two languages. MATLAB treats all numbers as doubles by default, so integers must be converted from double into a type recognised by MATLAB, the unsigned 64-bit integer (uint64), which C++ can also handle. As for the output, a sparse array object must be created in C++ to be fed to MATLAB as part of a list.

5 Results

The technique presented has been tested with a numerical example for the truss, CST and LST elements. The example used is a cantilever subjected to a set of edge loads applied at randomly-selected elements at the top boundary, each with a different random magnitude, as well as the structure's self-weight.

	span L [mm]	depth h [mm]	E [MPa]	Cross-section A [mm^2]
Value	2000	100	2e5	314.15

Table 1: Truss parameters.

	span L [mm]	depth h [mm]	E [MPa]	t [mm]	ν	Self-weight γ [N/m^3]
Value	2000	100	2e5	10	0.3	80e3

Table 2: CST parameters.

If the nodal displacements were known by the user, the global stiffness matrix assembler would be used instead and the nodal forces vector F would be obtained from multiplying K and U. Given that an initial nodal displacement vector is not available, a vector with the set of uniformly distributed loads q at the top elements is provided as input. This is then used to assemble the equivalent nodal forces vector together with the self-weight. Now, the assembled F and and an initial guess for the set of UDLs (q_0) are input into the Gauss-Newton function. The function assembles the Jacobian for the problem and a result for q_c is obtained. Finally, the error is calculated for the computed edge loads q_c by comparing them with the vector of input distributed loads q as follows:

$$error = \frac{|q - q_c|}{|q|} \times 100 \tag{41}$$

The maximum error in the system has been found for an increasingly larger number of domain divisions in the x-direction n_x which is equal to the length of the vector of edge loads, for both all three elements: The inverse analysis tool works as intended for CST and truss elements, finding the correct loading configuration for simple cases such as a UDL spanning the whole length of the beam-like structure, as well as for cases where random load values are assigned only at random locations along the top edge.

Interestingly, when applying this optimisation routine to certain two-dimensional truss structures, non-uniqueness issues arise (see Figure 5), that is, a solution to the minimisation problem which ensures that Equation 32 is satisfied is found, but the solution set does not correspond to the correct loading configuration. This occurs because the restrained DOFs are not solved for, which means only the equivalent nodal forces vectors are compared and several solutions that produce the original F_p vector exist. To avoid non-uniqueness, the linear system of equations must be solved at each iteration, hence obtaining the reaction forces, thus ensuring that only one set of UDLs exists. Systems comprised of CST elements did not present non-uniqueness issues.

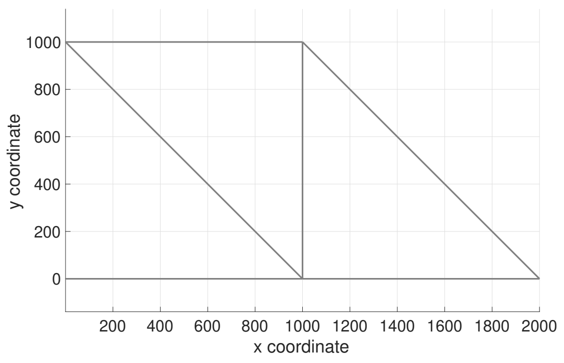

Figure 9: Example of ill-conditioned and non-unique truss problem. Vertical point loads are applied at the midpoint of all members, except the vertical one.

Another interesting observation is that the Gauss-Newton algorithm, regardless of different boundary conditions, number of elements or loading configurations, always converges after exactly 2 iterations to a low tolerance ($L2 - norm(R) < 10^{-6}$) for both elements when the problem is well-conditioned and not singular.

The type of forces are assumed to be uniform either across the whole edge of an element or throughout its surface. Whilst this is not likely to always be the case, for larger systems with a high number of elements, the edge loads will approach the real load distribution over the whole span, eventually making the error due to this assumption negligible. This is analogous to a numerical integration with the trapezoidal method being more accurate for a higher number of points.

6 Notes on the Code Developed

The code developed allows to the user to perform a linear analysis using the finite element method on rectangular beam-like structures built from either truss, CST or LST elements under an imposed set of UDLs at their upper boundary; and or to carry out a load reconstruction. The user can choose to have cantilever or simply-supported boundary conditions; to have a uniform or a random set of UDLs; and to include or neglect self-weight. Further, the user can decide to plot the mesh and its deformed shape.

Development has been carried out on a *macOS Mojave 10.14.16*, on *MATLAB˙R2021˙a* with an *Xcode 11.1* C++ compiler for the MEX functions. The global stiffness matrix assembler, as well as the solver for the nodal displacements have been validated using *ABAQUS 2020*. In addition, checks for the user input data have been implemented to avoid unintended run-time errors, whilst tests such as force balance or K symmetry have been devised where possible to guarantee accurate results.

The tool achieves machine-precision accuracy in load reconstruction for the truss and LST elements, and barely over that for the CST element.

As it can be see

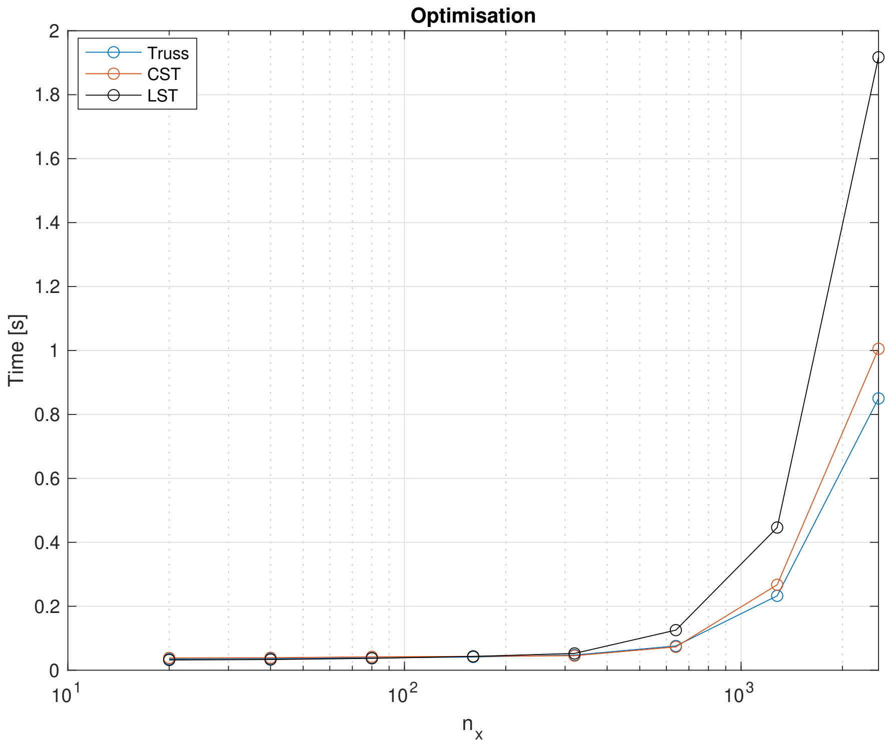

Figure 10: Time.

- SCALABILITY

7 Conclusions

The fact that this approach works for truss and CST elements leads to believe that it should also work for shell elements. Given that the Jacobian is only dependent on the set of element loads and not on the element formulation, this approach should be applicable to any type of finite element. Three-dimensional shell elements are cumbersome to formulate, but the approach presented here would allow to perform inverse analyses with high speed and accuracy, once a stiffness matrix assembler was completed.

Contrary to what was reported by Sanayei & Saletnik (1996), instances of non-uniqueness have been encountered for truss elements. Singular problems have been likewise observed for the former axial element and have been overcome with the Levenberg–Marquardt modification to the Gauss-Newton algorithm.

Nevertheless, the nodal displacement vector provided is assumed to be complete and accurate. Alas, in many instances a full knowledge of the displacement field is not available and displacement measurements are most likely inaccurate. These constraints introduce error into the external force estimation. Furthermore, displacement measurements require a frame of reference, which makes other variables such as strains more adequate for this type of inverse problems. In addition, the element loads considered are uniform along an edge or the whole surface of the element. This is not always accurate, even though for fine meshes this simplification results in an almost negligible error.

Current load reconstruction and shape sensing research efforts are focused on the implementation of the inverse Finite Element Method (iFEM) for a range of different structures, where an extensive library of elements is being developed (Tessler 2003), (Tessler & Spangler 2005), (Gherlone et al. 2014), (Savino et al. 2020). Future work on the subject of inverse analysis of 2D structures as well as 3D shell structures could perhaps consider combining the iFEM approach with the computation of the Jacobian of the equivalent nodal forces vector and the element edge and surface loads presented here.

References

Chock, J. M. & Kapania, R. K. (2003), 'Load updating for finite element models', *AIAA journal* **41**(9), 1667–1673.

Chong, E. K. & Zak, S. H. (2004), *An introduction to optimization*, John Wiley & Sons.

Cook, R. D. (1993), 'Further development of a three-node triangular shell element', *International journal for numerical methods in engineering* **36**(8), 1413–1425.

Cook, R. D., Malkus, D. S., Plesha, M. E. & Witt, R. J. (2007), *Concepts and Applications of Finite Element Analysis*, John Wiley & Sons.

GeoStru (n.d.), 'Constant strain triangle (t3)'.
URL: *https://help.geostru.eu/gfas/en/constantstraintriangle(t3).htm*

Gherlone, M., Cerracchio, P., Mattone, M., Di Sciuva, M. & Tessler, A. (2014), 'An inverse finite element method for beam shape sensing: theoretical framework and experimental validation', *Smart Materials and Structures* **23**(4), 045027.

Jin, L. (1994), Analysis and evaluation of a shell finite element with drilling degree of freedom, PhD thesis.

Li, J. & Kapania, R. K. (2007), 'Load updating for nonlinear finite element models', *AIAA journal* **45**(7), 1444–1458.

Macorini, L. (2019), *CI3-321 Computational Engineering Analysis. Lecture notes*, Imperial College London.

Sadowski, A. (2019), *CI9-STR-39 Theory of Shells. Lecture notes*, Imperial College London.

Sanayei, M. & Saletnik, M. J. (1996), 'Parameter estimation of structures from static strain measurements. i: Formulation', *Journal of Structural Engineering* **122**(5), 555–562.

Savino, P., Tondolo, F., Gherlone, M. & Tessler, A. (2020), 'Application of inverse finite element method to shape sensing of curved beams', *Sensors* **20**(24), 7012.

Streamliner, F. D. (n.d.), 'What is shear locking?'.
URL: *https://femds.com/FEM$_\mathrm{G}$uidelines/Prevent$_\mathrm{S}$hear$_\mathrm{L}$ocking*

Sun, E. Q. (2006), Shear locking and hourglassing in msc nastran, abaqus, and ansys, *in* 'MSc software users meeting', pp. 1–9.

Tessler, A. (2003), *A Variational Principle for Reconstruction of Elastic Deformations in Shear Deformable Plates and Shells*, NASA technical memorandum, National Aeronautics and Space Administration, Langley Research Center.
URL: *https://books.google.es/books?id=XYY9AQAAMAAJ*

Tessler, A. & Spangler, J. L. (2005), 'A least-squares variational method for full-field reconstruction of elastic deformations in shear-deformable plates and shells', *Computer methods in applied mechanics and engineering* **194**(2-5), 327–339.

A Tables

	$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$	$n_x = 1280$	$n_x = 2560$
$DOFs$	810	2898	10914	42306	166530	660738
$error$ [%]	0	0	0	0	0	0

Table 3: Results for cantilever built from truss elements.

	$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$	$n_x = 1280$	$n_x = 2560$
$DOFs$	810	2898	10914	42306	166530	660738
$error$ [%]	0	0	0	0	0	0

Table 4: Results for cantilever built from CST elements.

	$n_x = 80$	$n_x = 160$	$n_x = 320$	$n_x = 640$	$n_x = 1280$	$n_x = 2560$
$DOFs$	810	2898	10914	42306	166530	660738
$error$ [%]	0	0	0	0	0	0

Table 5: Results for cantilever built from LST elements.

B Figures

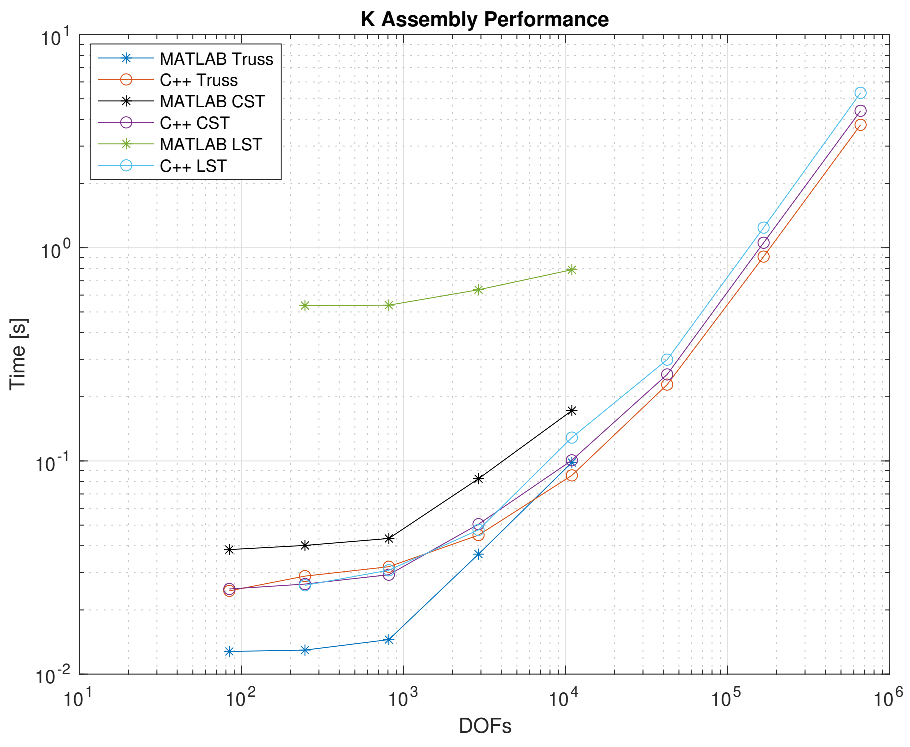

Figure 11: K assembly time.

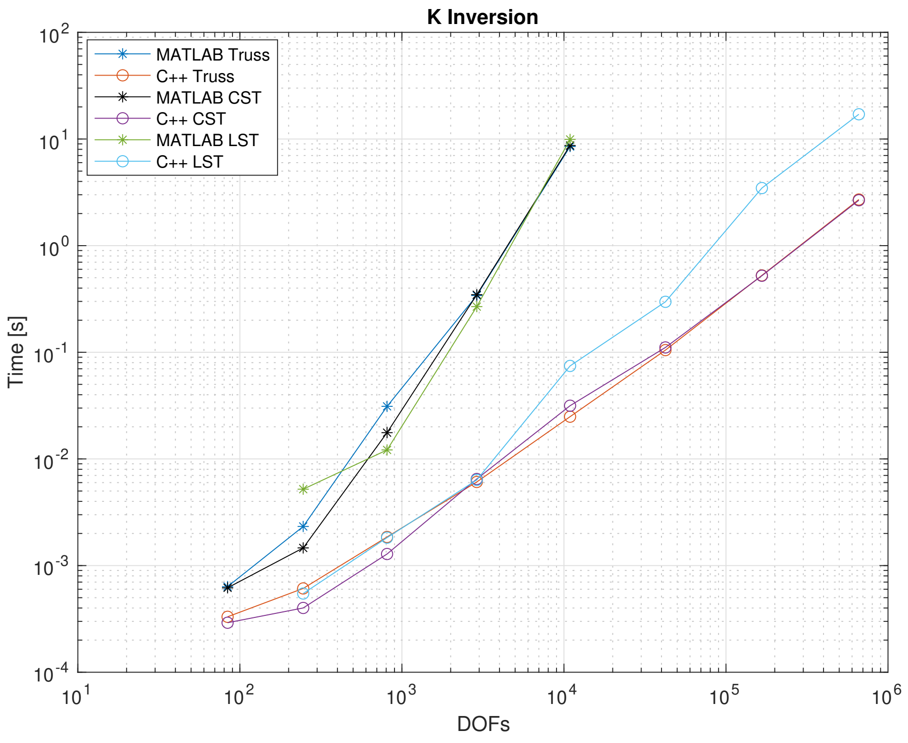

Figure 12: K inversion time.