Chair of Applied Mechanics
Department of Mechanical Engineering
Technical University of Munich

Dynamic Response of a 3D Structure

Project, Structural Dynamics (MW 2136) - SS17

Pablo Rodríguez Robles, Matr.-Nr. 03685205

Abstract

In this work, a simple Finite Element code is used to model a hangar structure (see Appendix A). Once this model is set up and tested, the dynamic response of the structure is studied. This study is further divided in two parts: First, the free vibration response of the system is analysed and eigenmodes and eigenfrequencies are computed by means of the Power Iteration method with Orthogonal Deflation. Secondly, the transient response of the structure is computed using a time-integration approach with the help of implicit and explicit Newmark schemes.

Contents

	Initial structure	Structure with new cross bars
First eigenfrequency (Hz)	0.2272	0.2616
Second eigenfrequency (Hz)	0.2628	0.2630
Third eigenfrequency (Hz)	0.3832	0.5390
Fourth eigenfrequency (Hz)	0.4651	0.5410

Table 1: First four eigenfrequencies of the original and modified structure.

1 Beam and Bar Finite Element Model

1.1 Building the Finite Element Model

The studied structure is a hangar modelled using different bar and beam elements in a Finite Element code. After defining each of the nodes of the hangar's geometry, the finite elements are created for different materials joining these nodes. Some boundary conditions, i.e. fixations at the structure's foundation, are imposed. For a complete description of the problem's geometry consult the Appendix A.

After studying the code and the hangar representation, it was decided to add some additional bar elements to the hangar (only for this section) and observe how the structure is stiffened, leading to an increase of its eigenfrequencies (code snippet of this implementation in Appendix B). As it can be seen in Figure 1, four new bar elements were added to the structure in the modes 1, 21, 30, 31, 101, 121, 130 and 131. Keeping everything the same (i.e. boundary conditions and rest of the elements) a free vibration test using the code included was conducted to observe how the hangar stiffness changed, see Table 1. The introduction of the new reinforcement elements increases the eigenfrequencies of the structure, which means that its stiffness is also increased. The new elements act as additional constraints between the degrees of freedom of the nodes, thus stiffening the system.

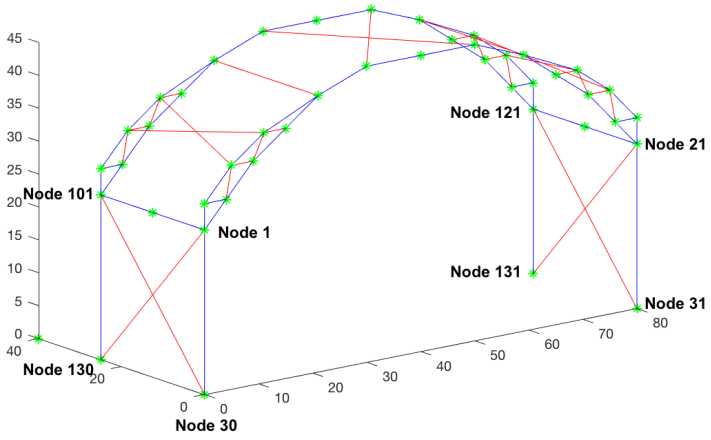

Figure 1: New cross bars added to stiffen the structure.

1.2 Checking the Model

It is possible to analyse the correct model implementation checking the rigid body modes of the structure. To do this, it is necessary to remove all the boundary conditions fixing some of the modes.

First we check that for a rigid body translation mode the model generates no force. For this purpose, the stiffness matrix $\boldsymbol{K}$ for the non-constrained system is used, i.e. the matrix $\boldsymbol{K}$ provided by the code before applying the boundary conditions. It must be satisfy

$$\boldsymbol{K}\boldsymbol{u}_{trans} = \boldsymbol{0} \tag{1}$$

where $\boldsymbol{u}_{trans}$ is a translation rigid body mode. For a model with n nodes, since every of the nodes has 6 DoF (degrees of freedom), the translation vector $\boldsymbol{u}_{trans}$ is a $(6n \times 1)$ vector with one of these patterns

$$\begin{aligned}
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}^T \\
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}^T \\
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & \dots \end{bmatrix}^T
\end{aligned}$$

or any linear combination of them[1]. When this operation is performed in a finite precision system like MATLAB, the result is not a vector containing all zeros, but containing very small values. Then, it is necessary to check if each element of the resulting array is zero up to a certain tolerance, which results to be true for the model presented. See Appendix C for the implementation.

In order to check the total mass of the structure a similar procedure can be applied. In this case the translation rigid body mode $\boldsymbol{u}_{trans}$ is limited to an unit amplitude. This is because we want to apply an unit acceleration of the structure, for a unit acceleration the inertial force produced is equal to the mass (Newton's second law). Then, the translation vector $\boldsymbol{u}_{trans}$ is a $(6n \times 1)$ vector with one of these three patterns

$$\begin{aligned}
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}^T \\
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & \dots \end{bmatrix}^T \\
\boldsymbol{K}\boldsymbol{u}_{trans} &= \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & \dots \end{bmatrix}^T
\end{aligned}$$

Here no combination or scaling is permitted[2]. The total mass is computed with

$$\boldsymbol{u}_{trans}^T \boldsymbol{M} \boldsymbol{u}_{trans} = m_{total} \tag{2}$$

The result is $15\,949\,\mathrm{kg}$ for all the cases tried (as expected). See also Appendix C for the implementation.

2 Numerical Methods for Dynamic Analysis

2.1 Free Vibration

Here the inverse iteration technique with deflation is implemented to compute the first eigenmodes and eigenfrequencies of the presented system. This implementation is available in the Appendix D. To conduct this analysis, as well as the following, all the boundary conditions fixing the base of the structure are again imposed.

[1] No only rigid translations are permitted, it would also be possible to use rigid rotations. But they are much more difficult to impose and give the same information.

[2] Actually it is possible to use a combination of accelerations in the three space directions if the modulus is one. However, the case was only checked for the simple cases assumed above.

Number	Eigenfrequency (Hz)	Difference w.r.t. `eig` (Hz)
1.	0.227 172 304 677 848	0.054 471 220 312 102e−7
2.	0.262 763 224 333 734	−0.080 858 777 451 454e−7
3.	0.383 225 170 876 377	0.006 938 190 577 621e−7
4.	0.465 050 647 525 288	−0.009 548 085 100 342e−7
5.	0.545 936 718 826 645	0.006 106 022 354 402e−7
6.	0.595 175 315 500 743	−0.005 646 566 547 668e−7
7.	0.647 241 555 567 568	−0.033 387 831 388 509e−7
8.	0.726 562 208 273 420	−0.017 445 920 263 981e−7
9.	0.808 302 304 799 511	−0.027 269 725 366 708e−7
10.	1.124 506 735 794 542	0.096 435 655 017 046e−7

Table 2: Eigenfrequencies for the first 10 eigenmodes.

The implementation of the inverse iteration follows the scheme proposed in the lecture notes [1], including the orthogonal deflation to eliminate the components on the previously computed eigenmodes.

Table 2 shows the result of the computation of the first 10 eigenfrequencies of the structure by means of the inverse iteration with orthogonal deflation, as well as a comparison with regard to the Matlab function `eig` as a difference of $\omega_{\text{inviter}} - \omega_{\text{eig}}$, for a convergence criterion of $\epsilon = 1\text{e}{-7}$.

When verifying the accuracy of the computations, it can be observed that it is related to the convergence criteria (see order of magnitude of the difference in Table 2). It cannot be observed that the accuracy of the higher modes is deteriorating due to the application of successive deflection.

To further investigate this behaviour, the author decided to compute up to the first 100 of eigenfrequencies and plot the absolute value of the difference between the to methods for different convergence criteria, see Figure 2. It can be observed how to up to a certain convergence criterion the solution starts to be accurate w.r.t. the Matlab `eig` solution, and their difference is related to the convergence criterion used (which is related to the accuracy of the solution of the inverse iteration method). But no significant deterioration can be observed for the higher frequencies in any case.

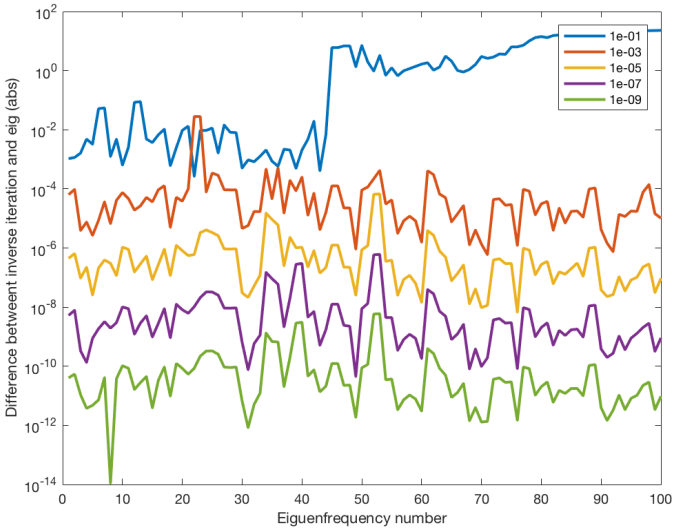

Figure 2: Difference between inverse iteration and eig for the first 100 eigenfrequencies and different convergence criteria.

From here we can conclude that the accuracy of the method does not deteriorate for higher frequencies, at least w.r.t. the result of Matlab.

It is important to note that as described in the lecture notes [1] the number of nearly converged eigenvalues is equal to half of the degrees of freedom in the model. However, the approximation obtained is an upper bound to the corresponding exact eigenvalues.

2.2 Transient Analysis

Here, the transient response of the hangar structure to a force applied on its frontal is computed using two different cases of the Newmark scheme. This force has the form of a triangular function, with an amplitude of $100\,\mathrm{N}$ and a period equal to half the period of the fourth eigenfrequency (Appendix E contains the implementation of this force).

2.2.1 Implicit Newmark Scheme

Implicit Newmark method was implemented, using the average constant acceleration scheme (i.e. $\gamma = \frac{1}{2}$ and $\beta = \frac{1}{4}$). For such an implicit method, stability is unconditional, what means that the stability of the method is not influenced by the time step chosen. Taking this into account, it was estimated that in order to represent the higher frequencies properly, the time resolution should be smaller than the period of the highest eigenfrequency[3]. Being $\omega = 355.36\,\mathrm{Hz}$ its value, a time step smaller than $0.0028\,\mathrm{s}$ would represent it properly.

For a decreasing time step the solution (see Figure 4) starts converging when using a time step of the order $1\mathrm{e}{-3}$, which is consistent with the assumption of a time step smaller than the period for the highest eigenfrequency.

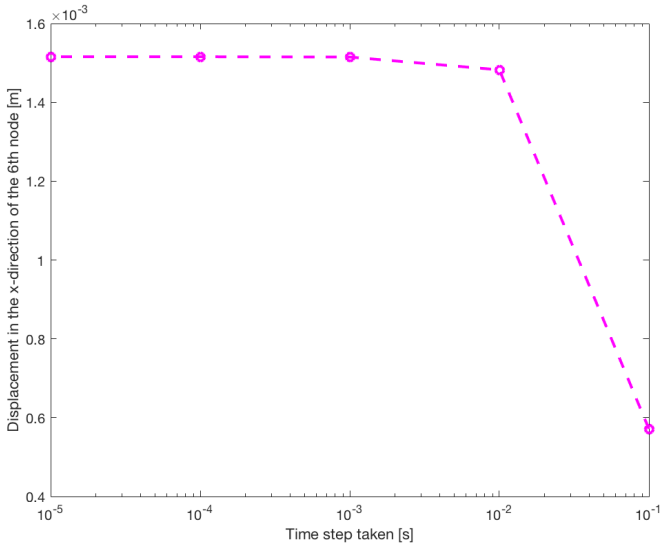

Figure 3: Convergence of the solution for displacement of node 6 in the x-direction for different time steps.

2.2.2 Explicit Newmark Scheme

Explicit Newmark method was implemented, using the central difference scheme (i.e. $\gamma = \frac{1}{2}$ and $\beta = 0$), which is explicit for the displacements.

The computational cost of the method was studied taking as a metric the execution time of the script

[3]Since the model has a finite number of degrees of freedom, it has also a finite number of eigenfrequencies, which where computed with Matlab's eig function.

containing its implementation (see Appendix G). When compared[4] to the implicit scheme implemented above using a time step of $1\mathrm{e}{-4}$ for both, propagating the solution in time until 30 times the period of the excitation force, the explicit scheme took $98.635\,\mathrm{s}$ and the implicit scheme $104.308\,\mathrm{s}$. This is not, of course, such a big difference and can be explained because the scheme implemented is not fully explicit (i.e. α is still not equal to zero), which means than the accelerations still need to be computed. The acceleration computation requires solving the following set of equations [1]

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{M} + h\gamma\boldsymbol{C} + h^2\beta\boldsymbol{K} \\ \boldsymbol{S}\ddot{\boldsymbol{q}}_{n+1} &= \boldsymbol{p}_{n+1} - \boldsymbol{C}\dot{\boldsymbol{q}}^*_{n+1} - \boldsymbol{K}\boldsymbol{q}^*_{n+1} \end{aligned} \tag{3}$$

Since there is no damping and $\beta = 0$, the first of the expressions above is simplified to $\boldsymbol{S} = \boldsymbol{M}$. This would be an advantage in case the system's model were using a lumped mass strategy, since the inversion of $\boldsymbol{S}$ in the second expression would be cost-free for a diagonal matrix $\boldsymbol{M}$. Since this is not the case, to compute the acceleration $\ddot{\boldsymbol{q}}_{n+1}$ is still necessary to solve a linear problem. Other difference for this scheme is that in the correction step (see Equation 4) the accelerations are not necessary to compute the new displacements, but still we need them to compute the velocities. Thus, there exist no noticeable performance improvement (i.e. the linear system of Equation 3 must yet be solved).

$$\begin{aligned} \dot{\boldsymbol{q}}_{n+1} &= \dot{\boldsymbol{q}}^*_{n+1} + h\gamma\ddot{\boldsymbol{q}}_{n+1} \\ \boldsymbol{q}_{n+1} &= \boldsymbol{q}^*_{n+1} + h^2\beta\ddot{\boldsymbol{q}}_{n+1} \end{aligned} \tag{4}$$

One must conclude that the only reason to use this scheme is in presence of a lumped mass matrix (i.e. diagonal) and when a small time step is required (since the method is not unconditionally stable) because there is an special interest in the higher frequencies (impact response or wave propagation).

Concerning the stability of the method, in order to satisfy the stability condition the time step h must be chosen such as $\omega h \leq 2$. Where ω is the highest eigenfrequency contained in the model. Following this statement, for the system studied the time step should be

$$\begin{aligned} h &\leq \frac{2}{\omega} \\ h &\leq \frac{2}{355.365\,\mathrm{Hz}} \\ h &\leq 0.0056\,\mathrm{s} \end{aligned}$$

After several numerical experiments (i.e. variations of the time step and observing if the solution explodes) it was observed that the computations become unstable for a time step bigger than $0.000\,895\,72\,\mathrm{s}$, which corresponds to a highest eigenfrequency of the order of $2200\,\mathrm{Hz}$. The author found no explanation to this behaviour, since although it has been stated that the computation of the last half of the eigenfrequencies is not accurate, they act as an upper bound to the real eigenfrequencies of the structure. What is more, the frequency used in the definition of the stability criterion should correspond to the highest frequency that the model can achieve (which was accurately computed if compared with the Matlab solution) and not the frequency of the real physical system.

When the solution (in the plot for the displacements of mode 6) of both methods is compared for the same time step of $0.0001\,\mathrm{s}$, they seem to give the same result.

[4] Running on a MacBook Pro (Retina, 13-inch, Early 2015) with a 2.7 GHz Intel Core i5 processor and 8GB or RAM. Using Matlab R2016a, 64-bit (maci64) from February 11, 2016.

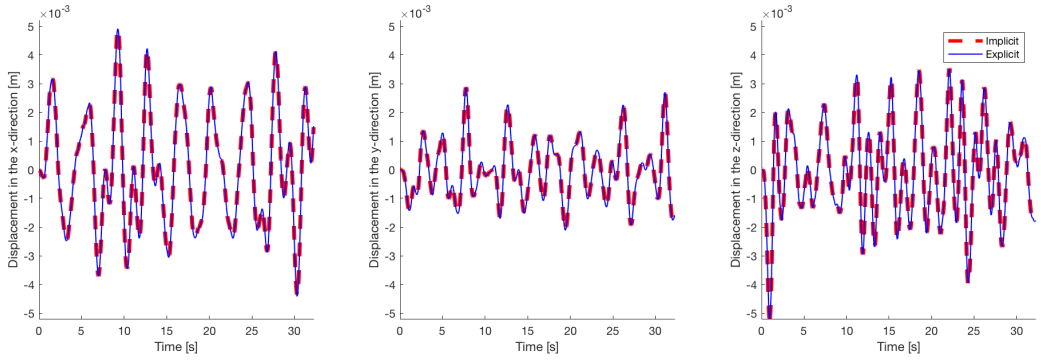

Figure 4: Comparison of implicit and explicit methods solution fo r the displacements of mode 6 with a time step of 0.0001 s.

A Hangar Structure Description

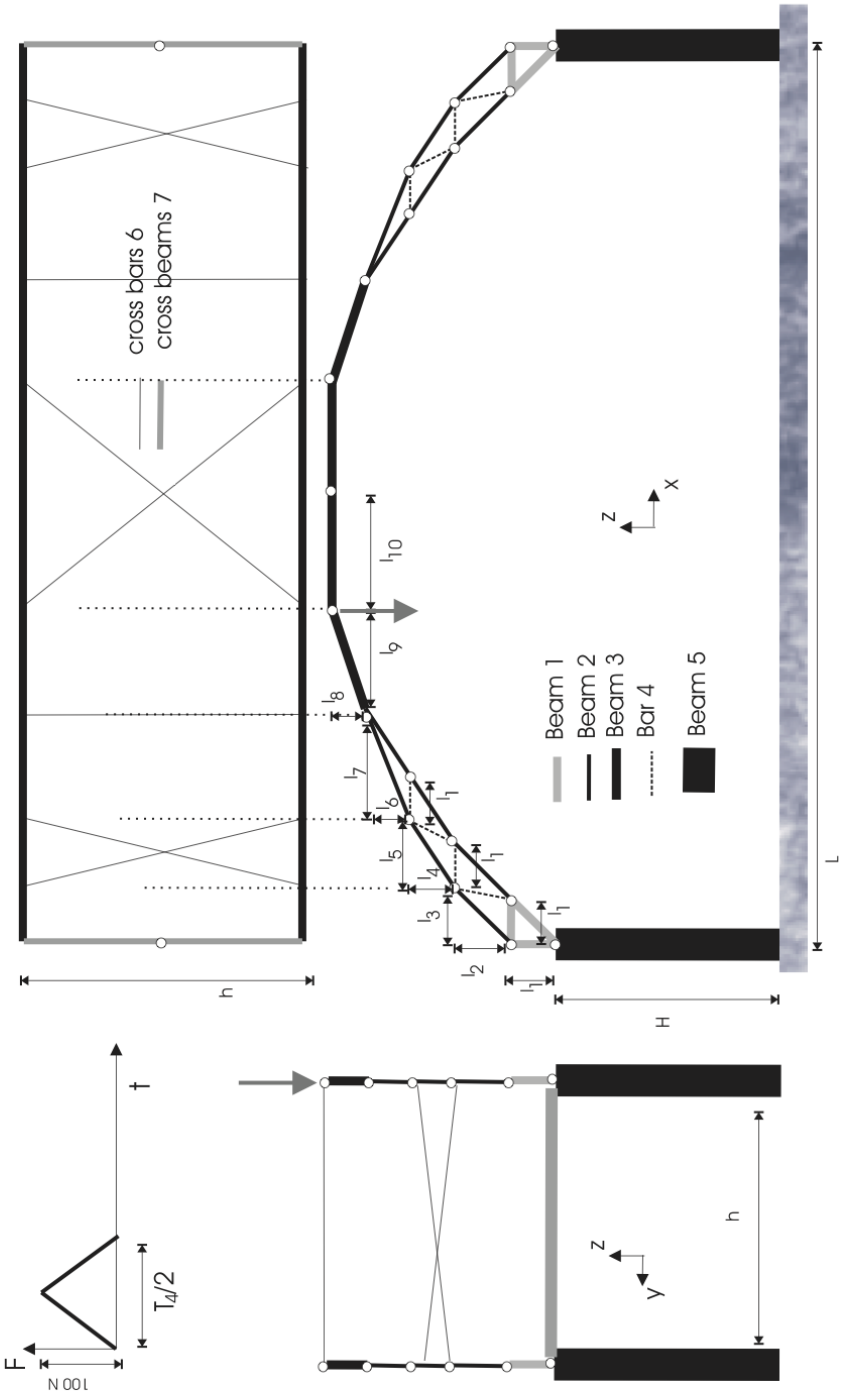

Figure 5: Hangar structure.

B Building the Finite Element Model Code

```
%% Model elements

% Add additional cross bars
Elements(77,:) = [1    6        1  130   200];
Elements(78,:) = [1    6       30  101   200];
Elements(79,:) = [1    6       21  131   200];
Elements(80,:) = [1    6       31  121   200];
```

C Model Check Code

```
%% Check the model: Stiffness matrix K
% Using K before applying the boundary conditions

% Get dimesions of u form K
[m, n] = size(K);

% Construct rigid body mode
u = zeros(n, 1);
u(1:6:n, 1) = 1;
u(2:6:n, 1) = 1;
u(3:6:n, 1) = 1;

% Tolerance for almost 0
eps = 1e-7;

% Ku = 0 (for almost equal, fininte precision).
% If true Ku = 0 holds
any(abs(K * u) > eps)

%% Check the model: Total mass
% Using M before applying the boundary conditions

% Get dimesions of u form M
[m, n] = size(M);

% Construct rigid body mode
u = zeros(n, 1);
u(1:6:n, 1) = 0;
u(2:6:n, 1) = 0;
u(3:6:n, 1) = 1;

% u'Mu = m_total
% Returns 1.5949e+04 kg for all cases
u' * M * u
```

D Inverse Iteration Method with Orthogonal Deflation Code

```
%% Compute eigensolutions: Inverse Iteration Method

% Make K and M full
```

```
K = full(K);
M = full(M);

% Check K is not singular using rank
[m, n] = size(K);
if rank(K) < m
error('K is not singular')
end

% Tolerance for convergence check using Rayleight quotients
eps = 1e-7;

% Number of eigenvalues computed
neig = 10;

% Matrix containing the orthonomalization operators.
% Inintially identity, when a new eigenmode is found
% the new operator is included using matrix multiplication:
% P = I*P1*P2*P3...Pk
P = eye(size(M));

% Preallocation of vector containing the
% eigenfrequencies of the system
omega2 = zeros(neig, 1);

% Preallocation of vector containing the
% eigenfrequencies of the system
eigmodes = zeros(m, neig);

% Althoug it is more efficient to use directly
% Matlab's backslash operator, here LU is used to
% look at the outputs.
[L,U] = lu(K);

for j=1:neig

% Arbitrary starting vector
z0 = rand(n, 1);
z = z0;

% Rayleight quotient
rayquo = (z' * K * z) ./ (z' * M * z);

% Previous Rayleight quotient
rayquo_0 = 0;

% Convergence condition,
% 1 meaning 'True'
condition = 1;

% Used to count number of iterations carried
iter = 0;
```

```
while condition

% Removes components of the already computed
% eigenmodes from the starting vector
z = P * z;

% Partial solution
y = M * z;

% New iterate z
% LUz = Lb = y, Uz = b
b = L\y;
z = U\b;

% Normalized iterate
z = z / norm(z);

% Rayleight quotient
rayquo_0 = rayquo;
rayquo = (z' * K * z) / (z' * M * z);

% Convergence condition.
% If greater than the tolerance keep iterating
condition = abs(rayquo - rayquo_0) > eps;

iter = iter + 1;
end

% Eigenmode
x = z;

% Add normalisation operator to the rest of them
P = P * (eye(size(M)) - (x * x' * M) / (x' * M * x));

% Add computed eigenfrequency
omega2(j) = rayquo;

% Add computed eigenmode
eigmodes(:, j) = x;
end

% The threshold used as tolerance for the inverse power influences
% the difference betweeen the Matlab computations and the computations from
% the code.

% Show more decimal places
format long

omega2(1:neig);
eigenfreq_inverse = sqrt(omega2)/(2*pi)
```

E Transient Force Code

```
function [ F ] = force( t, Ndof, locnod, dof_rem )
% Force applied on the hangar.
% Its time variation is a triangular function af total
% length equal to half the period of the fourth eigenfrequency
% and has an amplitude of 100 N.

% Half the period of the fourth eigenfrequency
T = 0.5 * (1 / 0.4651);

if (t < 0) || (t > T)
f = 0;

elseif t < T / 2
f = 100 / (T / 2) * t;

else
f = 200 - 100 / (T / 2) * t;

end

% Ndof, number of degrees of freedom after applying boundary conditions
F = zeros(Ndof, 1);

% Force is applied vertically and pointing downwards on Node 9
loc = locnod(9, 3);

F(loc) = - f;

F = F(dof_rem);

end
```

F Implicit Newmark Scheme Code

```
%% Transient dynamics: Implicit Newmark scheme

% Newmark scheme parameters
% (Averaged constant acceleration)
gamma = 1/2;
beta = 1/4;

% Make K and M full
K = full(K);
M = full(M);

% Dimensions of M (and K) after BCs
[m, n] = size(M);

% Half the period of the fourth eigenfrequency
T = 0.5 * (1 / 0.4651);

% Time boundaries [s]
t0 = 0;
```

```
tf = 30 * T;

% Time step [s]
step = [1e-1 1e-2 1e-3 1e-4 1e-5];
sol = zeros(length(step), 1);

for j=1:length(step)

h = step(j);

% Time [s]
time = t0:h:tf;

[~, s] = size(time);

% Vector of displacements
q_0 = zeros(m, 1);
q = zeros(m, s);
q(:, 1) = q_0;

% Vector of velocities
q_dot0 = zeros(m, 1);
q_dot = zeros(m, s);
q_dot(:, 1) = q_dot0;

% Vector of accelerations
% Case 1, initial acceleration equals zero
q_ddot0 = zeros(m, 1);
q_ddot = zeros(m, s);
q_ddot(:, 1) = q_ddot0;

% Vector of accelerations
% Case 2, compute initital acceleration
% (In this case not necessary)

% External force at time t0
% p = force(t0, Ndof, locnod, dof_rem);

% q_ddot0 = M \ (p - K * q_0)

% For a constant time step h, S does not change.
% Its factorization can be reutilized
S = M + h^2 * beta * K;
[L,U] = lu(S);


for i = 1:s-1

% Prediction
q_dotstar = q_dot(:, i) + (1 - gamma) .* h .* q_ddot(:, i);
q_star = q(:, i) + h .* q_dot(:, i) + ...
(0.5 - beta) .* h.^2 .* q_ddot(:, i);
```

```
% Acceleration
p = force(time(i+1), Ndof, locnod, dof_rem);

% LU q_ddot = L b = (p - K * q_star), U q_ddot = b
b = L \ (p - K * q_star);
q_ddot(:, i+1) = U \ b;

% Correction
q_dot(:, i+1) = q_dotstar + h .* gamma .* q_ddot(:, i+1);
q(:, i+1) = q_star + h.^2 .* beta .* q_ddot(:, i+1);

end

% Save x-direction displacement of the 6th node
sol(j) = q(31 , end);
end
```

G Explicit Newmark Scheme Code

```
%% Transient dynamics: Explicit Newmark scheme

% Newmark scheme parameters
% (Averaged constant acceleration)
gamma = 1/2;
beta = 0;

% Make K and M full
K = full(K);
M = full(M);

% Dimensions of M (and K) after BCs
[m, n] = size(M);

% Half the period of the fourth eigenfrequency
T = 0.5 * (1 / 0.4651);

% Time boundaries [s]
t0 = 0;
tf = 30 * T;

% Time step [s]
step = [1e-4];
sol = zeros(length(step), 1);

for j=1:length(step)

h = step(j);

limit = 355.3646 * h;
if limit > 2;
disp('Unstable')
else
disp('Stable')
```

```
end

% Time [s]
time = t0:h:tf;

[~, s] = size(time);

% Vector of displacements
q_0 = zeros(m, 1);
q = zeros(m, s);
q(:, 1) = q_0;

% Vector of velocities
q_dot0 = zeros(m, 1);
q_dot = zeros(m, s);
q_dot(:, 1) = q_dot0;

% Vector of accelerations
% Case 1, initial acceleration equals zero
q_ddot0 = zeros(m, 1);
q_ddot = zeros(m, s);
q_ddot(:, 1) = q_ddot0;

% Vector of accelerations
% Case 2, compute initital acceleration
% (In this case not necessary)

% External force at time t0
% p = force(t0, Ndof, locnod, dof_rem);

% q_ddot0 = M \ (p - K * q_0)

% For a constant time step h, S does not change.
% Its factorization can be reutilized
S = M;
[L,U] = lu(S);


for i = 1:s-1

% Prediction
q_dotstar = q_dot(:, i) + (1 - gamma) .* h .* q_ddot(:, i);
q_star = q(:, i) + h .* q_dot(:, i) + ...
0.5 .* h.^2 .* q_ddot(:, i);

% Acceleration
p = force(time(i+1), Ndof, locnod, dof_rem);

% LU q_ddot = L b = (p - K * q_star), U q_ddot = b
b = L \ (p - K * q_star);
q_ddot(:, i+1) = U \ b;

% Correction
```

```
q_dot(:, i+1) = q_dotstar + h .* gamma .* q_ddot(:, i+1);
q(:, i+1) = q_star;

end

% Save x-direction displacement of the 6th node
sol(j) = q(31 , end);

end
```

References

[1] D. J. Rixen, *Lecture notes on Structural Dynamics*. Technische Universität München.