3-D Beam Finite Element Programming - A Practical Guide Part 2 – Dynamic Modal Analysis (Software Included)

July 2024

Paolo Varagnolo: freelance engineer - Italy, info@studioingegneriavaragnolo.com
Private Practice

Contents

1 Introduction

After having presented the static analysis features of the finite element program MdFem (Mono dimensional Finite Element Method) in [9], in this paper one type of dynamic analysis is described. As for the static analysis, here too the main reference is the procedures described by K. J. Bathe in [1].

The programming language of MdFem is vb.net by Microsoft, instead of FORTRAN as in [1].

The purpose of this work is to provide a perfectly working program, with some practical tips on the adopted techniques, with detailed, step by step explanations of some examples. The aim is to provide a contribution to those who want to approach the finite element method (FEM), from the programmer point of view.

The theoretical and mathematical framework will not be addressed, since it is already widely available in many books and papers.

The MdFem program implements a 3-D Beam element. It is a straight, 2 nodes element: at each node there are 3 translational and 3 rotational degrees of freedom (dof). This element is capable of transmitting axial and shear forces, along with torque and bending moments.

There are two main approaches in dynamic analysis: 1) Modal Response Spectrum Analysis; 2) Direct step by step Integration. Both methods are contemplated by European normative [7], but the first is simpler and mostly used, therefore this is the method described below.

Modal Response Spectrum Analysis implies the following steps:

1) eigenvalues and eigenvectors solution;
2) modal participation factors and participating masses calculation;
3) definition of the response spectrum (spectra);
4) calculation of seismic forces from spectrum accelerations.

In the following, the four topics will be described.

2 Eigenvalues and eigenvectors solution

The method used in this paper is the Subspace Iteration Method, developed by K. J. Bathe and fully described in [1]. The original program published in [1] has been translated in vb.net language: the routines involved in the method have been first tested with some examples, and then they have been added to the static program published in [9].

“The basic objective of the subspace iteration method is to solve for the lowest p eigenvalues and corresponding eigenvectors satisfying”

$$[K]\,[\phi] = [M]\,[\phi][\Lambda]$$

where:

$[K]$ is the global stiffness matrix of the structure (here stored in compacted form);

$[\phi]$ is the matrix containing the eigenvectors $[\phi_1, \ldots\ldots, \phi_p]$;

$[M]$ is the mass matrix (also stored in compacted form);

$[\Lambda]$ is a diagonal matrix containing the eigenvalues $\lambda_1, \ldots\ldots, \lambda_p$.

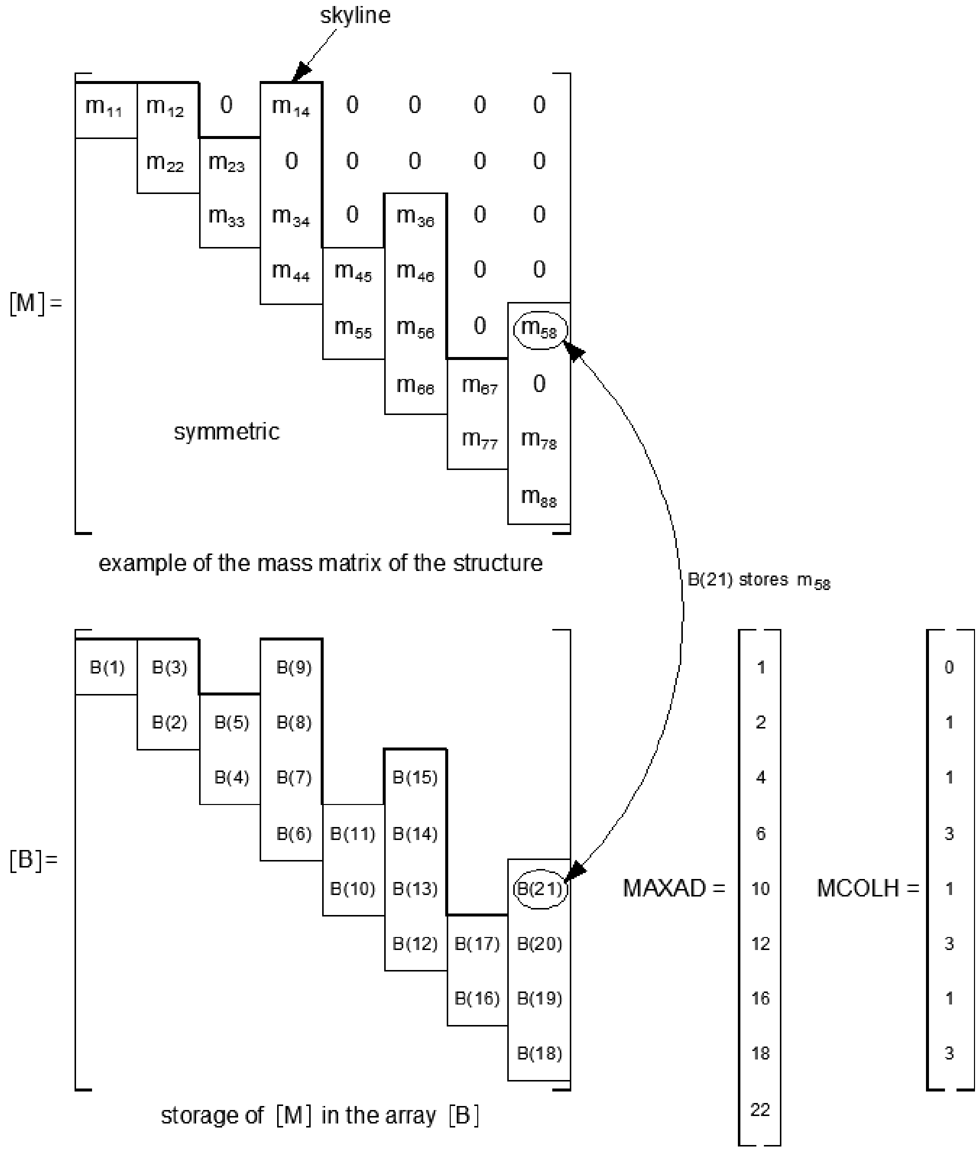
The storage of matrices in compacted form has been explained in [9] but is briefly showed in the next sub section.

For the structural problems considered in this article the eigenvalues λ_i are the free vibration frequencies squared, ω_i^2. The following expressions therefore apply:

- circular frequencies $\omega_i = \sqrt{\lambda_i}$ (rad/s)
- natural frequencies $\nu_i = \omega_i/(2\pi)$
- periods $T_i = 1/\nu_i$

2.1 Storage scheme of the global stiffness and mass matrices

The global stiffness matrix $[K]$ and the mass matrix $[M]$ of the structure are stored in compacted form with the active columns scheme as described in [1]. Only the part of the matrix below the skyline and including the diagonal is stored in a one-dimensional array $[A]$ for the stiffness matrix and $[B]$ for the mass matrix. In order to know the addresses of the $[K]$, $[M]$ elements in $[A]$, $[B]$, the positions of the diagonal elements are stored in the array ***MAXAD***(), and the number of non-zero elements above the diagonal element is stored in the array ***MCOLH***(). The following figure shows the storage scheme of the global mass matrix in the array $[B]$.



In the program MdFem the total number of degrees of freedom of the structure in called ***NDOFT,*** while the total number of elements below the skyline is called ***NKGLO*** for the stiffness matrix and ***NMGLO*** for the mass matrix.

In the MdFem program, array $[A]$ is called ***GLOBK***(*NKGLO*) and array $[B]$ is called ***GLOBM***(*NMGLO*).

2.2 Consistent and lumped mass matrix

To represent the mass distribution of a structure in a finite element model, the mass of an element is proportionally applied to its nodes. There are two methods to convert the element mass into a matrix referred to the degrees of freedom of the nodes: the mass matrix can be consistent or lumped.

Consistent mass matrix

The consistent mass matrix M_e of an element is calculated with the same shape functions used for the calculation of the stiffness matrix, with the expression:

$$M_e = \int_{V_e} [N]^T \rho\, [N]\, dV$$

where:

$[N]$ = element shape function matrix

ρ = material mass density

V_e = element volume

The program MdFem uses the following closed form expression provided by Katsikadelis in [6].

$$M_e = \begin{bmatrix} m_{jj} & m_{jk} \\ m_{kj} & m_{kk} \end{bmatrix}$$

where:

$$m_{jj} = \frac{m_e}{420}\begin{bmatrix} 140 & 0 & 0 & 0 & 0 & 0 \\ 0 & 156 & 0 & 0 & 0 & 22L \\ 0 & 0 & 156 & 0 & -22L & 0 \\ 0 & 0 & 0 & 140r_G^2 & 0 & 0 \\ 0 & 0 & -22L & 0 & 4L^2 & 0 \\ 0 & 22L & 0 & 0 & 0 & 4L^2 \end{bmatrix}$$

$$m_{kk} = \frac{m_e}{420}\begin{bmatrix} 140 & 0 & 0 & 0 & 0 & 0 \\ 0 & 156 & 0 & 0 & 0 & -22L \\ 0 & 0 & 156 & 0 & 22L & 0 \\ 0 & 0 & 0 & 140r_G^2 & 0 & 0 \\ 0 & 0 & 22L & 0 & 4L^2 & 0 \\ 0 & -22L & 0 & 0 & 0 & 4L^2 \end{bmatrix}$$

$$m_{kj} = \frac{m_e}{420}\begin{bmatrix} 70 & 0 & 0 & 0 & 0 & 0 \\ 0 & 54 & 0 & 0 & 0 & 13L \\ 0 & 0 & 54 & 0 & -13L & 0 \\ 0 & 0 & 0 & 70r_G^2 & 0 & 0 \\ 0 & 0 & 13L & 0 & -3L^2 & 0 \\ 0 & -13L & 0 & 0 & 0 & -3L^2 \end{bmatrix}$$

$$m_{jk} = \left(m_{kj}\right)^T$$

Where:

$m_e = \rho A L$ is the total mass of element e;

$r_G = \sqrt{J_{T0}/A}$ is the radius of gyration of the cross-section;

Lumped mass matrix

The lumped mass matrix is simpler: in this approach the mass is lumped equally onto the nodes, and is associated only to the translational degrees of freedom. The diagonal elements of the mass matrix related to the translational degrees of freedom are the reactions of a simply supported beam.

The lumped mass matrix is:

$$m_{jj} = \begin{bmatrix} m_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_1 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$m_{kk} = \begin{bmatrix} m_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & m_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & m_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$m_{jk} = m_{kj} = 0$$

Where:

$m_1 = m_2 = \rho AL$ / 2 = half of the element mass

2.3 Examples of the Subspace Iteration Method

The program `Sspace` from [1], translated in vb.net language, is listed in Appendix A. In the following some examples are presented, in order to verify the correctness of the new implementation.

<u>Example 12.1 from "Finite Element Procedures in Engineering Analysis" [1]</u>

The data reported in the reference are:

$$K = \begin{bmatrix} 2 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 2 \end{bmatrix} \quad M = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0.5 \end{bmatrix}$$

The input data for the program are listed below.

TITLE: example 12.1 Bathe NDOFT 3 NKGLO 5 NMGLO 3 NROOT 2 MAXAD 1 2 4 6	global stiffness matrix in compacted form 2.0E+00 4.0E+00 -1.0E+00 2.0E+00 -1.0E+00 global mass matrix in compacted form 0.5E+00 1.0E+00 0.5E+00

The following table shows the results published in [1], calculated by hand with 9 iterations, and those calculated by the program `Sspace()`.

Eigenvalues		Eigenvectors			
[1]	Sspace	[1]	Sspace	[1]	Sspace
2	2	0.7057	0.70711	1.001	1.0
4	4	0.7071	0.70711	0.001	3.77E-15
		0.7085	0.70711	-0.999	-1.0

Example 12.3 from "Finite Element Procedures in Engineering Analysis" [1]

The data reported in the reference are:

$$K = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad M = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The input data for the program are listed below.

```
TITLE: example 12.3 Bathe
NDOFN
    4
NKGLO
    7
NMGLO
    4
NROOT
    2
MAXAD
    1
    2
    4
    6
    8
```

```
global stiffness matrix in compacted form
        2.0E+00
        2.0E+00
       -1.0E+00
        2.0E+00
       -1.0E+00
        1.0E+00
       -1.0E+00
global mass matrix in compacted form
        0.0E+00
        2.0E+00
        0.0E+00
        1.0E+00
```

The following table shows the results published in [1], calculated by hand with 1 iteration, and those calculated by the program `Sspace()`.

Eigenvalues		Eigenvectors			
[1]	Sspace	[1]	Sspace	[1]	Sspace
0.14645	0.14645	0.25	0.25	0.25	0.25
0.085355	0.085355	0.50	0.50	0.50	0.50
		0.60355	0.60355	0.10355	-0.10355
		0.70711	0.70711	0.70711	-0.70711

It must be noted that, if there are some zero masses in a diagonal (lumped) mass matrix, only one iteration is needed with the subspace iteration method, as established in [1].

Example 1 from “Programma di calcolo SSPACE” [2]

The data reported in the reference are:

$$K = \begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad M = \begin{bmatrix} 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The input data for the program are listed below.

TITLE: example 1 Boreggio Benà NDOFN 4 NKGLO 7 NMGLO 4 NROOT 4 MAXAD 1 2 4 6 8	global stiffness matrix in compacted form 2.E+00 2.E+00 -1.E+00 2.E+00 -1.E+00 1.E+00 -1.E+00 global mass matrix in compacted form 3.E+00 2.E+00 4.E+00 1.E+00

The following table shows the results published in [2], and those calculated by the program `Sspace()`.

Eigenvalues	
[2]	Sspace
0.0517654	0.051765
0.464639	0.46464
1.1714	1.1714
1.47886	1.4789

Eigenvectors							
[2]	Sspace	[2]	Sspace	[2]	Sspace	[2]	Sspace
0.1548	0.15483	0.4634	0.46336	-0.2471	-0.24706	0.1834	0.18336
0.2856	0.28561	0.2808	0.28083	0.3741	0.37410	-0.4468	-0.44678
0.3868	0.38682	-0.1627	-0.16266	0.1188	0.11882	0.2445	0.24452
0.4079	0.40794	-0.3038	-0.30384	-0.6932	-0.69319	-0.5106	-0.51064

The results listed in [2] have been checked with the program CAL78, written by professor Edward L. Wilson of the University of California, Berkeley.

Example 2 from “Programma di calcolo SSPACE” [2]

The data reported in the reference are:

$$K = \begin{bmatrix} 5 & -4 & 1 & 0 \\ -4 & 6 & -4 & 1 \\ 1 & -4 & 6 & -4 \\ 0 & 1 & -4 & 5 \end{bmatrix} \quad M = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The input data for the program are listed below.

TITLE: example 2 Boreggio Benà NDOFN 4 NKGLO 9 NMGLO 4 NROOT 4 MAXAD 1 2 4 7 10	global stiffness matrix in compacted form 5.E+00 6.E+00 -4.E+00 6.E+00 -4.E+00 1.E+00 5.E+00 -4.E+00 1.E+00 global mass matrix in compacted form 2.E+00 2.E+00 1.E+00 1.E+00

The following table shows the results published in [2], and those calculated by the program `Sspace()`.

Eigenvalues	
[2]	Sspace
0.0965373	0.096537
1.39147	1.3915
4.37355	4.3735
10.6384	10.638

Eigenvectors							
[2]	Sspace	[2]	Sspace	[2]	Sspace	[2]	Sspace
0.3126	0.31263	0.4453	0.44527	-0.4387	-0.43867	-0.1076	-0.10756
0.4955	0.49548	0.1244	0.12444	0.4167	0.41674	0.2556	0.25563
0.4791	0.47912	-0.4894	-0.48944	0.02322	0.023222	-0.7283	-0.72825
0.2898	0.28979	-0.5770	-0.57702	-0.5170	-0.51697	0.5620	0.56197

The results listed in [2] have been checked with the program CAL78, written by professor Edward L. Wilson of the University of California, Berkeley.

The differences are only due to the different significant digits used to write the values.

3 Modal participation factors and participating masses

The **participation factor** $[\Gamma_j]$ for mode j, in the direction i, indicates how strongly motion in the generalized directions 1÷6 (x, y, z, xx, yy, zz) is represented in the eigenvector of this mode. It is defined as:

$$[\Gamma_j] = \frac{[\phi_j]^T [M] [r_i]}{[\phi_j]^T [M] [\phi_j]} \tag{3.1}$$

where:

$[\phi_j]$ is the eigenvector j

$[M]$ is the global mass matrix of the structure

$[r_i]$ is the influence matrix which represents the displacements of the masses resulting from static application of a unit ground displacement. The influence vector induces a rigid body motion in all modes.

$[\phi_j]^T [M] [\phi_j]$ is the generalized mass m$_\alpha$

Eigenvalues are only determined up to an arbitrary scalar factor, i.e. if $[\phi_j]$ is an eigenvector, also c $[\phi_j]$ is an eigenvector (c is a real number). Eigenvectors are usually normalized such that its norm equals 1 (normalized to unity), or such that $[\phi_j]^T [M] [\phi_j]$ = 1 (normalized to the mass matrix).

The eigenvectors calculated by the program presented in this paper are normalized to the mass matrix.

The influence matrix is an identity matrix for the translational degrees of freedom (dof), while contains the differences of the displacements with respect to a centre of rotation for the rotational dof. In the present treatment only the participation factors and the participating masses for the translational dof will be considered.

With these assumptions expression (3.1) becomes:

$$[\Gamma_j] = [\phi_j]^T [M] \tag{3.2}$$

The **participating mass** for the j mode is calculated with the following expression:

$$[M_j] = \frac{[\Gamma_j]^2}{m_{tot}}$$

Where:

m_{tot} is the total mass of the structure.

Italian building code (NTC 2018 §7.3.3.1) prescribes that the number of calculated modes must involve at least 85% of the total mass in each of the x, y directions; other codes request a slightly different percentage.

The routine that calculates participation factors and participating masses is presented below.

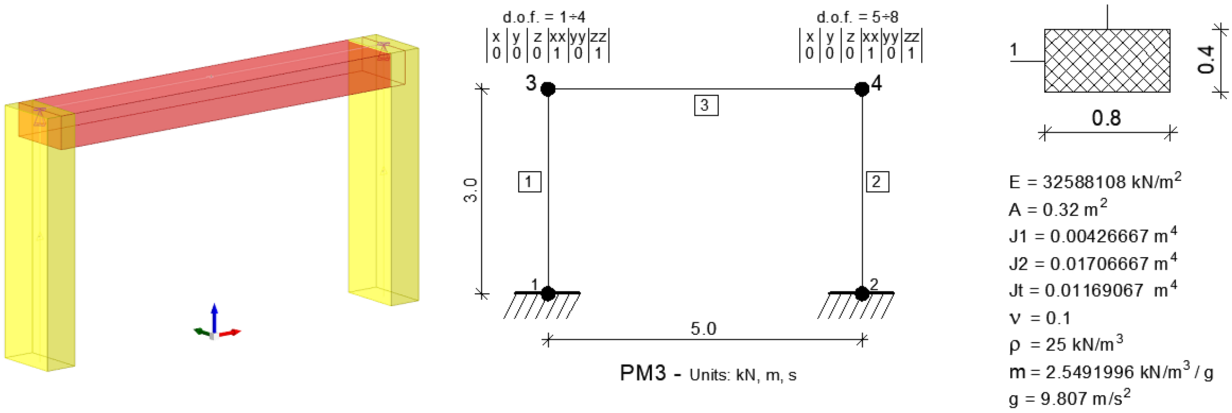
Matrix ***EigenVec****(NDOFT, NROOT)* contains the eigenvectors calculated in Sub Sspace, where *NROOT* is the number of requested modes.

Vector ***TotalMass****(NDOFT)* is calculated in *Sub* ***ADDBAN_Mass*** and in *Sub* ***AddLoadMass*** listed in Appendix B.

```
Sub ParticipantMassesCalculation(EigenVec(,) As Double)

    'calculate participation factors pfj and participant masses pmj in each mode j

    'given: EigenVec(NDOFT, NROOT) calculated eigenvectors
    '       TotalMass(Idofn) = total mass previously calculated

    Dim Icoun, Jcoun, Ibase, Idire, Mass(NDOFT) As Integer
    Dim Product(NROOT, NDOFT), Sum As Double
    ReDim PartFact(NROOT, NDOFN), PartMas(NROOT, NDOFN)

    'find positions of (x, y, z) masses in the global mass matrix
    Icoun = 0
    For Ipoin = 1 To Npoin
        For Idofn = 1 To NDOFN
            If IDDOF(Idofn, Ipoin) > 0 Then
                Icoun += 1
                If Icoun <= NDOFT Then 'And Idofn <= 3 Then
                    Mass(Icoun) = Idofn
                End If
            End If
        Next Idofn
    Next Ipoin

    'calculation of Modal Participation Factors
    If IfLumped Then
        'product of eigenvectors (transpose) by lumped mass matrix: FTxM
        For Iroot = 1 To NROOT
            For Idofn = 1 To NDOFT
                Product(Iroot, Idofn) = EigenVec(Idofn, Iroot) * GLOBM(Idofn)
            Next Idofn
        Next Iroot
    Else
        'product of eigenvectors (transpose) by consistent mass matrix: FTxM
        For Iroot = 1 To NROOT
            For Idofn = 1 To NDOFT
                Sum = 0

                'calculate number of elements above the skyline
                Icoun = Idofn - MCOLH(Idofn) - 1

                'loop on elements above the diagonal and under the skyline
                For i% = MAXAD(Idofn) + MCOLH(Idofn) To MAXAD(Idofn) Step -1
                    Icoun += 1
                    Sum += EigenVec(Icoun, Iroot) * GLOBM(i%)
                Next i%
                'loop on the elements under the diagonal (= elements on the row)
                Jcoun = 0
                For i% = Idofn + 1 To NDOFT
                    Icoun += 1
                    Jcoun += 1
                    Ibase = MAXAD(i%)
                    If MCOLH(i%) >= Jcoun Then
                        Sum += EigenVec(Icoun, Iroot) * GLOBM(Ibase + Jcoun)
                    End If
                Next i%
                Product(Iroot, Idofn) = Sum
            Next Idofn
        Next Iroot
        'Call Control_FTxMxF(Product, EigenVec) 'must be the Identity matrix: FTxMxF = I
    End If

    'calculate Modal Participation Factors in (x, y, z, rx, ry, rz) directions
    For Iroot = 1 To NROOT
        For Idofn = 1 To NDOFT
            Idire = Mass(Idofn)
            PartFact(Iroot, Idire) += Product(Iroot, Idofn)
        Next Idofn
```

```
        Next Iroot

        'calculation of Modal Participating masses
        For Iroot = 1 To NROOT
            For Idime = 1 To NDIME
                If TotalMass(Idime) <> 0 Then
                    PartMas(Iroot, Idime) = PartFact(Iroot, Idime) ^ 2 / TotalMass(Idime)
                End If
            Next Idime
        Next Iroot

    End Sub
```

Follows a detailed calculation of participating masses as performed by the previous routine, referring to Example 2, whose data are shown in the next figure.



There are no concentrated or distributed loads, therefore the total mass m_{TOT} on not restrained dof is:

$$m_{TOT} = 2.5491996 \times 0.32 \times (2 \times 1.5 + 5) = 6.525951$$

From the figure we see that the total number of degrees of freedom *NDOFT* = 8, while the positions of (x, y, z) masses in array *Mass(1÷NDOFT)* are:

dof disp/rot	1 x-disp	2 y-disp	3 z-disp	4 yy-rot	5 x-disp	6 y-disp	7 z-disp	8 yy-rot
Mass(Idofn)	1	2	3	0	1	2	3	0

Next calculation is the product of $[\phi_j]^T[M]$ (rows by columns). Follows the $[\phi_j]^T$ matrix:

	PM3 calculated eigenvectors (normalized to the mass matrix)							
mode 1	3.91E-01	5.96E-20	1.99E-03	1.03E-01	3.91E-01	6.06E-20	-1.99E-03	1.03E-01
mode 2	-8.07E-17	3.91E-01	-4.10E-19	-2.13E-17	-8.07E-17	3.91E-01	4.10E-19	-2.13E-17
mode 3	-3.49E-16	-3.91E-01	8.78E-18	-9.23E-17	-3.49E-16	3.91E-01	1.23E-17	-9.22E-17
mode 4	1.50E-17	-2.95E-13	3.91E-01	-1.56E-15	1.41E-17	2.95E-13	3.91E-01	-1.56E-15
mode 5	-1.99E-03	8.08E-28	3.91E-01	7.36E-02	-1.99E-03	4.04E-28	-3.91E-01	7.36E-02
mode 6	-3.91E-01	9.22E-14	3.46E-18	-1.51E-01	3.91E-01	-9.59E-14	-1.86E-18	1.51E-01
direction	1	2	3	0	1	2	3	0
	x	y	z		x	y	z	

each row contains the eigenvector calculated for each vibration mode

each column contains the contributions of the modes to a single dof

The example here reported deals with the lumped mass matrix, that is listed here under (empty values are zeros):

Lumped global Mass Matrix

3.26E+00							
	3.26E+00						
		3.26E+00					
			0				
				3.26E+00			
					3.26E+00		
						3.26E+00	
							0

The product row by columns gives the following matrix:

	multiplication: eigenvalues x mass matrix							
mode 1	1.28E+00	1.94E-19	6.50E-03	0.00E+00	1.28E+00	1.98E-19	-6.50E-03	0.00E+00
mode 2	-2.63E-16	1.28E+00	-1.34E-18	0.00E+00	-2.63E-16	1.28E+00	1.34E-18	0.00E+00
mode 3	-1.14E-15	-1.28E+00	2.86E-17	0.00E+00	-1.14E-15	1.28E+00	4.02E-17	0.00E+00
mode 4	4.91E-17	-9.64E-13	1.28E+00	0.00E+00	4.60E-17	9.64E-13	1.28E+00	0.00E+00
mode 5	-6.50E-03	2.64E-27	1.28E+00	0.00E+00	-6.50E-03	1.32E-27	-1.28E+00	0.00E+00
mode 6	-1.28E+00	3.01E-13	1.13E-17	0.00E+00	1.28E+00	-3.13E-13	-6.08E-18	0.00E+00
direction	1	2	3	0	1	2	3	0
	x	y	z		x	y	z	

The multiplication gives the participation factors contributions split on the degrees of freedom of the FEM model. Adding the corresponding contributions (col. 1 + col. 5, col. 2 + col. 6, col. 3 + col. 7, col. 4 + col. 8) the final participation factors are obtained, as listed in the next table. On the right the values calculated by Sap4 are listed.

Participation Factors Γ_j (j=1÷6)			
2.55E+00	3.92E-19	0.00E+00	0.00E+00
-5.27E-16	2.55E+00	0.00E+00	0.00E+00
-2.28E-15	0.00E+00	6.89E-17	0.00E+00
9.51E-17	0.00E+00	2.55E+00	0.00E+00
-1.30E-02	3.95E-27	0.00E+00	0.00E+00
0.00E+00	-1.19E-14	5.21E-18	0.00E+00
1	2	3	0
x	y	z	yy-rot

```
Sap4 results

MODAL  PARTICIPATION  FACTORS
MODE    X-DIRECTION    Y-DIRECTION     Z-DIRECTION

   1    -0.2554E+01     0.8318E-20     0.5204E-17
   2     0.3133E-18    -0.2554E+01     0.4688E-14
   3     0.3236E-16    -0.9304E-10     0.4772E-12
   4    -0.8074E-18    -0.1006E-26    -0.2554E+01
   5     0.2149E-01     0.1681E-20    -0.3108E-14
```

Finally, the participating masses are calculated as participation factors squared, divided by the total mass. The result is listed below.

Participating masses = Γ_j^2 / M_{tot}		
1.00E+00	2.36E-38	0.00E+00
4.25E-32	1.00E+00	0.00E+00
7.96E-31	0.00E+00	7.27E-34
1.38E-33	0.00E+00	1.00E+00
2.59E-05	2.40E-54	0.00E+00
0.00E+00	2.18E-29	4.16E-36
1	2	3
x	y	z

Next images and tables refer to the same structure, calculated with consistent mass matrices.

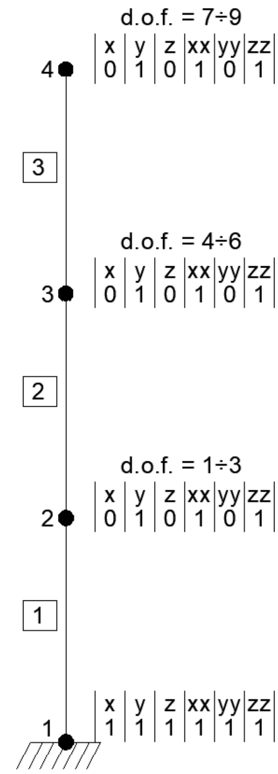
	Calculated eigenvectors with consistent mass matrix (normalized to mass matrix)							
	4.24E-01	8.08E-19	1.88E-03	1.08E-01	4.24E-01	-7.06E-19	-1.88E-03	1.08E-01
	5.03E-17	4.12E-01	3.53E-15	-5.45E-14	1.32E-15	4.12E-01	3.53E-15	5.49E-14
Eig^T	5.62E-03	2.24E-13	-3.13E-02	4.84E-01	-5.62E-03	1.18E-14	-3.13E-02	-4.84E-01
(NROOTxNDOFT)	1.76E-14	-5.13E-01	-1.23E-15	2.46E-14	1.71E-14	5.13E-01	-1.39E-15	-1.59E-14
	8.76E-02	-1.22E-14	-1.16E-01	9.63E-01	8.76E-02	1.22E-14	1.16E-01	9.63E-01
	-3.57E-01	-3.41E-13	3.95E-01	3.05E-01	3.57E-01	3.41E-13	3.95E-01	-3.05E-01
direction	1	2	3	0	1	2	3	0
	x	y	z		x	y	z	

	Consistent global Mass Matrix							
	2.268545	0	0	-0.38456	0.679787	0	0	0
	0	2.423925	0	0	0	0.524407	0	0
M	0	0	2.330697	-1.06824	0	0	0.524407	0.63123
(NDOFTxNDOFT)	-0.38456	0	-1.06824	1.180886	0	0	-0.63123	-0.72834
	0.679787	0	0	0	2.268545	0	0	-0.38456
	0	0.524407	0	0	0	2.423925	0	0
	0	0	0.524407	-0.63123	0	0	2.330697	1.068236
	0	0	0.63123	-0.72834	-0.38456	0	1.068236	1.180886

	multiplication: eigenvalues x mass matrix							
	1.21E+00	1.59E-18	-4.36E-02	-1.15E-01	1.21E+00	-1.29E-18	4.36E-02	-1.15E-01
	2.20E-14	1.21E+00	1.03E-13	-1.10E-13	-1.81E-14	1.21E+00	1.03E-13	1.10E-13
Prod	-1.77E-01	5.50E-13	-9.12E-01	9.76E-01	1.77E-01	1.46E-13	-9.12E-01	-9.76E-01
(NROOTxNDOFT)	4.21E-14	-9.75E-01	-3.99E-14	3.61E-14	5.69E-14	9.75E-01	-3.64E-14	-4.55E-14
	-1.12E-01	-2.33E-14	-6.31E-01	4.53E-01	-1.12E-01	2.33E-14	6.31E-01	4.53E-01
	-6.84E-01	-6.47E-13	6.10E-01	4.73E-02	6.84E-01	6.47E-13	6.10E-01	-4.73E-02

Participation Factors Γ_j (j=1÷6)					**Participating masses = Γ_j^2 / M_{tot}**		
2.42E+00	3.00E-19	0.00E+00	-2.30E-01		8.95E-01	1.38E-38	0.00E+00
3.90E-15	2.43E+00	2.06E-13	-3.72E-16		2.33E-30	9.04E-01	6.51E-27
0.00E+00	6.96E-13	-1.82E+00	0.00E+00		0.00E+00	7.43E-26	5.10E-01
9.90E-14	0.00E+00	-7.63E-14	-9.45E-15		1.50E-27	0.00E+00	8.92E-28
-2.24E-01	0.00E+00	0.00E+00	9.06E-01		7.70E-03	0.00E+00	0.00E+00
0.00E+00	0.00E+00	1.22E+00	0.00E+00		0.00E+00	0.00E+00	2.28E-01
1	2	3	0		1	2	3
x	y	z	yy-rot		x	y	z

In the routine ***ParticipantMassesCalculation***, after the multiplication $[\phi_j]^T[M]$, there is a commented line that hides a call to the routine ***Control_FTxMxF****(Product, EigenVec)*: some (positive) controls have been made in order to verify that the product $[\phi_j]^T[M]\ [\phi]$ gives the Identity matrix $[I]$.



Example

When a consistent mass matrix is adopted, the product $[\phi_j]^T[M]$ is particular, due to the storage scheme of the mass matrix (see § 2.1). The image on the left shows an example structure, useful to understand the procedure adopted.

There is a total number of 9 degrees of freedom: the tables below show the mass matrix ***GLOBM****(NMGLO)*, the first eigenvector transposed, vector ***MAXAD****(NDOFT+1)* and vector ***MCOLH****(NDOFT)*.

The mass matrix is symmetric, so the product of the eigenvector by the columns of the matrix follows the columns until it reaches the diagonal element and then follow the row, from the diagonal element to the right.

Furthermore, only the elements below the skyline are stored in the one-dimensional ***GLOBM***(*NMGLO*) array: for this reason, the procedure must evaluate the degrees of freedom involved, in order to associate the correct dof in the eigenvector with the correspondent dof of the mass matrix.

Starting with the columns, the multiplication must skip a number of elements equal to:

```
[Icoun = Idofn – MCOLH(Idofn) – 1]
```

Proceeding then with the rows, the multiplication is performed only if ***MCOLH****(current dof)* is greater than the current dof.

Consistent Mass Matrix

dof →	1	2	3	4	5	6	7	8	9	← dof
	0.325 1	0.000 3	-0.137 6	0.042 10	0.000 15	-0.046 21	/	/	/	1
		0.292 2	0.000	0.000	0.055	0.000	/	/	/	2
			0.356 4	0.046	0.000	-0.047	/	/	/	3
				0.203 7	0.000	-0.043	0.028 25	0.000 30	-0.020 36	4
					0.182 11	0.000	0.000	0.036	0.000	5
						0.082 16	0.020	0.000	-0.014	6
			symmetric				0.081 22	0.000	-0.034	7
								0.073 26	0.000	8
									0.019 31	9

First Eigenvector

dof →	1	2	3	4	5	6	7	8	9
	0.649	0.000	0.148	1.388	0.000	0.174	1.913	0.000	0.176

MAXAD(NDOFT+1)

dof →	1	2	3	4	5	6	7	8	9	
	1	2	4	7	11	16	22	26	31	37

MCOLH(NDOFT)

dof →	1	2	3	4	5	6	7	8	9
	0	1	2	3	4	5	3	4	5

When the eigenvector is multiplied by the 2nd dof, calling E() the eigenvector and M() the array containing the mass matrix, the multiplication is:

E(1)xM(3)+ E(2)xM(2)+ E(3)xM(5)+ E(4)xM(9)+ E(5)xM(14)+ E(6)xM(20) = 0

And, when the eigenvector is multiplied by the 7th dof, the multiplication is:

E(4)xM(25)+ E(5)xM(24)+ E(6)xM(23)+ E(7)xM(22)+ E(8)xM(27)+ E(9)xM(33) = 0.191

Let us see in detail how the program performs the multiplication, referring to the first eigenvector, for Idofn = 2.

```
Iroot = 1
Idofn = 2
      Sum = 0
      'calculate number of elements above the skyline
      Icoun = Idofn − MCOLH(Idofn) − 1 = 2 − 1 − 1 = 0
      'loop on elements above the diagonal and under the skyline
      For i% = MAXAD(2) + MCOLH(2) To MAXAD(2) Step −1 = 2 + 1 = 3 To 2 Step −1
            i% = 3:      Icoun += 1 = 1
                         Sum += EigenVec(1, 1) * GLOBM(3) = 0.649 * 0 = 0
            i% = 2:      Icoun += 1 = 2
                         Sum += EigenVec(2, 1) * GLOBM(2) = 0 * 0.292 = 0
      Next i%
      'loop on the elements under the diagonal (= elements on the row)
      Jcoun = 0
      For i% = Idofn + 1 To NDOFT = 2 + 1 = 3 To 9
            i% = 3:      Icoun += 1 = 3
                         Jcoun += 1 = 1
                         Ibase = MAXAD(i%) = 4
                         If MCOLH(3) = 2 >= Jcoun = 1 (True) Then
                               Sum += EigenVec(3, 1) * GLOBM(5) =
                                     0.  + 0.148 * 0 = 0 + 0 = 0
                         End If
            i% = 4:      Icoun += 1 = 4
                         Jcoun += 1 = 2
                         Ibase = MAXAD(i%) = 7
                         If MCOLH(4) = 3 >= Jcoun = 2 (True) Then
                               Sum += EigenVec(4, 1) * GLOBM(9) =
                                     0.  + 1.388 * 0 = 0 + 0 = 0
                         End If
            i% = 5:      Icoun += 1 = 5
                         Jcoun += 1 = 3
                         Ibase = MAXAD(i%) = 11
                         If MCOLH(5) = 4 >= Jcoun = 3 (True) Then
                               Sum += EigenVec(5, 1) * GLOBM(14) =
                                     0.  + 0 * 0.055 = 0 + 0 = 0
                         End If
            i% = 6:      Icoun += 1 = 6
                         Jcoun += 1 = 4
                         Ibase = MAXAD(i%) = 16
                         If MCOLH(6) = 5 >= Jcoun = 4 (True) Then
                               Sum += EigenVec(6, 1) * GLOBM(20) =
                                     0 + 0.0.174 * 0 = 0 + 0 = 0
                         End If
            i% = 7:      Icoun += 1 = 7
                         Jcoun += 1 = 5
                         Ibase = MAXAD(i%) = 22
                         If MCOLH(7) = 3 >= Jcoun = 5 (False) Then
                         End If
```

```
        i% = 8:      Icoun += 1 = 8
                     Jcoun += 1 = 6
                     Ibase = MAXAD(i%) = 26
                     If MCOLH(8) = 4 >= Jcoun = 6 (False) Then
                     End If
        i% = 9:      Icoun += 1 = 9
                     Jcoun += 1 = 7
                     Ibase = MAXAD(i%) = 31
                     If MCOLH(9) = 5 >= Jcoun = 7 (False) Then
                     End If
```

The product of the first eigenvector by the 2nd column of the mass matrix gives a zero value, meaning that the participation factor of the first vibration mode on the 2nd dof (z displacement at node 2) is null.

Let us see now in detail the multiplication, referring again to the first eigenvector, for Idofn = 7.

```
Iroot = 1
Idofn = 7
    Sum = 0
    'calculate number of elements above the skyline
    Icoun = Idofn - MCOLH(Idofn) - 1 = 7 - 3 - 1 = 3
    'loop on elements above the diagonal and under the skyline
    For i% = MAXAD(7) + MCOLH(7) To MAXAD(7) Step -1 = 22 + 3 = 25 To 22 Step -1
        i% = 25:     Icoun += 1 = 4
                     Sum += EigenVec(4, 1) * GLOBM(25) = 1.388 * 0.028 = 0.039
        i% = 24:     Icoun += 1 = 5
                     Sum += EigenVec(5, 1) * GLOBM(24) = 0.039 + 0 * 0 = 0.039
        i% = 23:     Icoun += 1 = 6
                     Sum += EigenVec(6, 1) * GLOBM(23) =
                         = 0.039 + 0.174 * 0.020 = 0.039 = 0.042
        i% = 22:     Icoun += 1 = 7
                     Sum += EigenVec(7, 1) * GLOBM(22) =
                         = 0.042 + 1.913 * 0.081 = 0.039 = 0.197
    Next i%
    'loop on the elements under the diagonal (= elements on the row)
    Jcoun = 0
    For i% = Idofn + 1 To NDOFT = 7 + 1 = 8 To 9
        i% = 8:      Icoun += 1 = 8
                     Jcoun += 1 = 1
                     Ibase = MAXAD(i%) = 26
                     If MCOLH(8) = 4 >= Jcoun = 1 (True) Then
                         Sum += EigenVec(8, 1) * GLOBM(27) =
                             = 0.197 + 0 * 0 = 0.197
                     End If
        i% = 9:      Icoun += 1 = 9
                     Jcoun += 1 = 2
                     Ibase = MAXAD(i%) = 31
                     If MCOLH(9) = 5 >= Jcoun = 2 (True) Then
                         Sum += EigenVec(9, 1) * GLOBM(33) =
                             0.197 + 0.176 * -0.034 = 0.197 + 0 = 0.191
                     End If
```

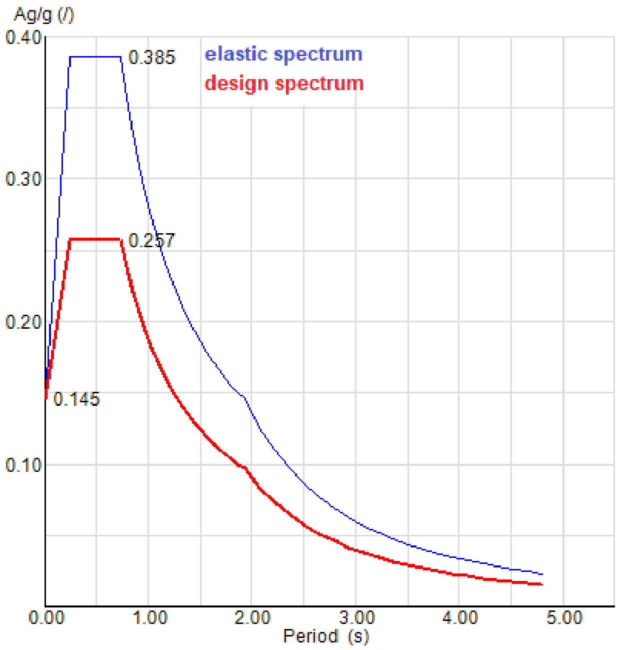
The participation factor of the first vibration mode on the 7th dof (x displacement at node 4) is 0.191.

4 Definition of the response spectrum

The response spectrum is a function defining the acceleration of simple harmonic oscillators with respect to the period, when they are subjected to a transient event. The response spectrum is a function of the natural period or frequency of each oscillator and of several other parameters listed below.

The response spectrum can be used to study the response of any linear system, also with many degrees of freedom (multi-degree of freedom systems), given its natural frequencies of oscillation.

The spectrum can be defined applying the rules given by any national building code, as a function of the site, the type of soil, the damping of the structure, the topographical conditions, and the behaviour factor. It has the shape shown in the next figure. The design spectrum is derived from the elastic one, dividing the values by the behaviour factor, depending on the ductility of the structure.



In the program MdFem the response spectrum is stored in a two-dimensional array, directly read from the input file. The program reads only one parameter that can modify the spectrum, the behaviour factor.

The program reads also the value of the viscous damping, but only to obtain the CQC combinations described in the next paragraph.

5 Calculation of seismic forces

In the response spectrum analysis (RSA) the main idea is that, for each vibration period, the maximum acceleration of the structure may be calculated with the response spectrum (of many single dof pendulums).

The peak force on the pendulum oscillating mass is:

$$F_{max} = k\, y_{max} = m\, a_{max} \tag{5.1}$$

since:

$$\omega = \sqrt{\frac{k}{m}}$$

we can also write:

$$y_{max} = \frac{a_{max}}{\omega^2} \tag{5.2}$$

Expressions (5.1) and (5.2) can be called the Forces Method and the Displacements Method in Response Spectrum Analysis.

The distribution either of the forces or of the displacements in the structure, for mode j, is evaluated multiplying F_{max} or y_{max} by the eigenvector ϕ_j, obtaining:

$$[f_j] = \phi_j\, F_{max} \tag{5.3}$$

$$[u_j] = \phi_j\, y_{max} \tag{5.4}$$

From the calculated nodal displacements, the internal forces of each element are obtained multiplying the element stiffness matrix by the displacements vector.

The above expressions (5.3), (5.4) must be evaluated for all the calculated periods, and then the results must be combined.

The most common combination rules are the Complete Quadratic Combination (CQC) and the Square Root of Sum of Squares (SRSS). Italian building code impose the CQC combination, while Eurocode 8 for instance allows both the methods.

Some comments on the reliability of the methods can be found in [10].

5.1 SRSS combination

Referring to the calculated forces, displacements, or internal forces, as a generic vector $[v_j]$, the SRSS combination is:

$$[v] = \sqrt{\sum_{j=1}^{p} [v_j]^2}$$

The value of a single component v_k of the vector is:

$$v_k = \sqrt{\sum_{j=1}^{p} v_{kj}^2}$$

Index j runs from 1 to the number of calculated eigenvalues p.

5.2 CQC combination

For simplicity we can directly refer to a single component v_k of the generic vector $[v_j]$.

The combined value is:

$$v_k = \sqrt{\sum_{i=1}^{p} \sum_{j=1}^{p} v_{ki}\, \rho_{ij}\, v_{kj}}$$

For constant modal damping ξ, the cross-modal coefficients ρ_{ij} are calculated as:

$$\rho_{ij} = \frac{8\xi^2 {\beta_{ij}}^{3/2}}{(1+\beta_{ij})\left[(1-\beta_{ij})^2 + 4\xi^2\beta_{ij}\right]}$$

where:

$\beta_{ij} = T_j/T_i$ is the ratio between the inverse of the vibration periods i, j.

The routine that calculates the cross-modal coefficients ρ_{ij} is presented below.

```
    Sub RoijCalc()
        'calculation of C.Q.C. Roij correlation coefficients

        Dim Betaij As Double

        ReDim Roij(NROOT, NROOT)

        For Imode = 1 To NROOT
            For Jmode = 1 To NROOT
                'Betaij = Period(Jmode) / Period(Imode)
                Betaij = CircFreq(Imode) / CircFreq(Jmode)
                Roij(Imode, Jmode) = 8 * DampCsi ^ 2 * Betaij ^ 1.5
                Roij(Imode, Jmode) /= (1 + Betaij)
                Roij(Imode, Jmode) /= ((1 - Betaij) ^ 2 + 4 * DampCsi ^ 2 * Betaij)
            Next Jmode
        Next Imode

    End Sub
```

5.3 Displacements methods in RSA

The routine that calculates the displacements of expression (5.4) is presented below.

The code is quite simple, due to the many comment lines and the self-explaining variables names. In any case, a brief description of the operations is given.

The participation factors are simply summed in the three x, y, z directions in the case of CQC combination, while the sum is made on the absolute values in the case of SRSS combination.

The subroutine calculates the displacements of every mode referring to the simple oscillator, and then those of the structure multiplying the simple oscillator displacements by the eigenvectors.

Follow the CQC or SRSS combination of the displacements, depending on the boolean variable **IfCQC** value.

Finally, spectrum displacements are stored in the array ***SpecDisp****(Iroot, Ipoin, Idofn)*, and combined displacements are stored in the array ***Displ****(CurCase, Ipoin, Idofn) =****CombStruDisp****(Icoun)*

```
    Sub SpectralResponse(EigenVec(,) As Double)

        'calculate Modal Response Spectrum analysis

        'given: EigenVec(NDOFT, NROOT) calculated eigenvectors

        Dim Icoun, Ldofn As Integer
        Dim PartFac, Force, Displa(NROOT), StruDisp(NDOFT, NROOT) As Double
        Dim Sum2 As Double
        ReDim SpecDisp(NROOT, Npoin, NDOFN), CombStruDisp(NDOFT)

        Dim Dire(3) As Single
        Dire(1) = 1 : Dire(2) = 1 : Dire(3) = 1

        'compute displacements of the simple oscillator and then those of the structure
        For Iroot = 1 To NROOT
            PartFac = 0
            For Idime = 1 To NDIME
                If IfCQC Then
                    PartFac += PartFact(Iroot, Idime) * Dire(Idime)
                Else
                    PartFac += Math.Abs(PartFact(Iroot, Idime)) * Dire(Idime)
                End If
            Next Idime

            Force = PartFac * ModalAccelerationCalc(Iroot)
            Displa(Iroot) = Force / CircFreq(Iroot) ^ 2 'displacement of the simple oscillator

            'calculation of structure displacements from the simple oscillator displacement
            For Idofn = 1 To NDOFT
                StruDisp(Idofn, Iroot) = EigenVec(Idofn, Iroot) * Displa(Iroot)
            Next Idofn
        Next Iroot

        'calculation of CQC or SRSS combinations of displacements for every d.o.f.
        Call RoijCalc() 'compute C.Q.C. Roij correlation coefficients (always needed with forces
method)
        For Idofn = 1 To NDOFT
            Sum2 = 0
            If IfCQC Then
                'Complete Quadratic Combinations (C.Q.C.) of displacements
                For Jmode = 1 To NROOT
                    For Imode = 1 To NROOT
                        Sum2 += Roij(Imode, Jmode) * StruDisp(Idofn, Imode) * StruDisp(Idofn,
Jmode)
                    Next Imode
                Next Jmode
```

```
            Else
                'Square Root of Sum of Squares combinations (S.R.S.S.) of displacements
                For Imode = 1 To NROOT
                    Sum2 += StruDisp(Idofn, Imode) ^ 2
                Next Imode
            End If

            CombStruDisp(Idofn) = Math.Sqrt(Sum2)
        Next Idofn

        'store spectrum displacements in SpecDisp(,,) array
        For Iroot = 1 To NROOT
            For Ipoin = 1 To Npoin
                For Idofn = 1 To NDOFN
                    Ldofn = IDDOF(Idofn, Ipoin)
                    If Ldofn > 0 Then
                        SpecDisp(Iroot, Ipoin, Idofn) = StruDisp(Ldofn, Iroot)
                    End If
                Next Idofn
            Next Ipoin
        Next Iroot

        'store spectrum displacement combinations in index 0 of displacements array
        CurCase = 0
        Icoun = 0
        For Ipoin = 1 To Npoin
            For Idofn = 1 To NDOFN
                If IDDOF(Idofn, Ipoin) > 0 Then
                    Icoun += 1
                    Displ(CurCase, Ipoin, Idofn) = CombStruDisp(Icoun)
                End If
            Next Idofn
        Next Ipoin

    End Sub
```

The displacements of the nodes are used to calculate the internal element forces, with the following routine. A brief description of the operations is given also for this routine.

The first part refers to the transformation of the calculated global displacements in local coordinates for each element. For the details of these calculations refer to Part 1 – Static Analysis [9]. Multiplying the local element matrices by the local displacements, the internal element forces are obtained.

Also for the internal forces it is necessary to combine the contributes of the vibration modes, either with the CQC or with the SRSS method.

```
    Sub STRBER(Icase As Integer, Nele1 As Integer, Nele2 As Integer, FileWork3 As String)

        'STRESS CALCULATION FOR BEAM OR WINKLER ELEMENTS
        'in the case of Response Spectrum Analysis

        Dim Index, Idofn, Idof1, Ldofn As Integer
        Dim GlobDisp(), LocDisp(), LocLoa(), Force(,), Sum, Stres(NEVAB) As Double
        Dim Toler As Double = 0.0000000001

        Using sr As StreamReader = File.OpenText(FileWork3) 'stiffness LOCAL matrices file

            ' *** LOOP OVER ELEMENTS

            For Ielem = Nele1 To Nele2

                'read stiffness matrix
```

```
            For Icolu = 1 To 78
                Stiff(Icolu) = sr.ReadLine
            Next Icolu

            ReDim Force(NEVAB, NROOT)

            For Iroot = 1 To NROOT
                'calculate displacements LocDisp() and NOT equivalent forces LocLoa() in
local coordinates
                'transformation T matrices are calculated only once after reading data
                ReDim GlobDisp(NEVAB), LocDisp(NEVAB), LocLoa(NEVAB)

                For Idofn = 1 To NDOFN
                    Idof1 = Idofn + NDOFN
                    GlobDisp(Idofn) = SpecDisp(Iroot, Inc1(Ielem), Idofn)
                    GlobDisp(Idof1) = SpecDisp(Iroot, Inc2(Ielem), Idofn)
                Next Idofn

                For Idofn = 1 To NDOFN
                    Idof1 = Idofn + NDOFN
                    For Jdofn = 1 To NDOFN
                        Ldofn = Jdofn + NDOFN
                        LocDisp(Idofn) += Tmat(Ielem, Idofn, Jdofn) * GlobDisp(Jdofn)
                        LocDisp(Idof1) += Tmat(Ielem, Idofn, Jdofn) * GlobDisp(Ldofn)
                    Next Jdofn
                Next Idofn

                'calculate forces in local system: [K] u = f
                For Ievab = 1 To NEVAB
                    Sum = 0
                    For Jevab = 1 To NEVAB
                        Index = Kpos(Ievab, Jevab)
                        Sum += Stiff(Index) * LocDisp(Jevab)
                    Next Jevab
                    Force(Ievab, Iroot) = Sum
                Next Ievab
            Next Iroot

            ReDim Stres(NEVAB)
            For Ievab = 1 To NEVAB
                If IfCQC Then
                    'Complete Quadratic Combinations (C.Q.C.) of displacements
                    For Jmode = 1 To NROOT
                        For Imode = 1 To NROOT
                            Stres(Ievab) += Roij(Imode, Jmode) * Force(Ievab, Imode) *
Force(Ievab, Jmode)
                            'Stres(Ievab) += Roij(Imode, Jmode) * Math.Abs(Force(Ievab,
Imode)) * Math.Abs(Force(Ievab, Jmode))
                        Next Imode
                    Next Jmode
                Else
                    'Square Root of Sum of Squares (S.R.S.S.) combinations of displacements
                    For Iroot = 1 To NROOT
                        Stres(Ievab) += Force(Ievab, Iroot) ^ 2
                    Next Iroot
                End If
                Stres(Ievab) = Math.Sqrt(Stres(Ievab))
            Next Ievab

            For Ievab = 1 To NEVAB
                If Math.Abs(Stres(Ievab)) < Toler Then Stres(Ievab) = 0.0
            Next Ievab

            'store values in Stre1(,,), Stre2(,,) matrices
            For Idofn = 1 To NDOFN
                Stre1(Icase, Ielem, Idofn) = -Stres(Idofn)
                Stre2(Icase, Ielem, Idofn) = Stres(Idofn + NDOFN)
            Next Idofn
        Next Ielem
```

```
        End Using

    End Sub
```

5.4 Forces method in RSA

The routine that calculates the forces of expression (5.3) is presented below. For this method the choice was made to use only CQC combination: actually, in some case SRSS combination leads to bad results. Besides, the actual implementation is not completely satisfactory, though it works fine in many cases. These cases include structures with regularity in plan and in elevation, as shown in the application examples further on.

Anyway, this method needs more insights, that will be presented in the future.

Follow a brief description of the operations made by the routine.

For every vibration mode j, the spectral acceleration a_j is calculated: then, at any node i, and for the x, y, z directions the program calculates the product $F_{ji} = m_i \, \phi_{ji} \, \gamma_j \, a_j$.

The next operation, performed in the second main loop on the vibration modes, is to transform the node forces in “storey” shears.

Follows the CQC combination of shear forces and the, finally, the calculation of “storey” forces as difference of “storey“ shears.

```
    Sub SeismicForcesCalculationCQC()
        'calculation of seismic forces with Forces Method

        Dim Disp As Double

        Dim   SeismicForce(NROOT,   Npoin,   NDIME),   ShearForces(NROOT,   Npoin,   NDIME),
BaseShearForce(NROOT, NDIME) As Double
        ReDim ModalAcc(NROOT)
        ReDim SeismicForceCQC(Npoin, NDIME)
        ReDim ShearForcesCQC(Npoin, NDIME)

        For Imode = 1 To NROOT
            ModalAcc(Imode) = ModalAccelerationCalc(Imode)
            For Ipoin = 1 To Npoin
                For Idime = 1 To NDIME
                    Disp = Displ(Ncase + NCOMB + Imode, Ipoin, Idime)
                    SeismicForce(Imode,   Ipoin,   Idime)   =   NodalMass(Ipoin)   *   Disp   *
PartFact(Imode, Idime) * ModalAcc(Imode)
                    BaseShearForce(Imode, Idime) += SeismicForce(Imode, Ipoin, Idime)
                Next Idime
            Next Ipoin
        Next Imode

        'transform seismic forces in storey shears
        For Imode = 1 To NROOT
            For Idime = 1 To NDIME
                ShearForces(Imode, Npoin, Idime) = SeismicForce(Imode, Npoin, Idime)
                For Ipoin = Npoin - 1 To 1 Step -1
                    ShearForces(Imode, Ipoin, Idime) = ShearForces(Imode, Ipoin + 1, Idime) +
SeismicForce(Imode, Ipoin, Idime)
                Next Ipoin
            Next Idime
        Next Imode

        'complete quadratic combinations C.Q.C. of shear forces (D.M. 14.01.2018)
        For Jmode = 1 To NROOT
            For Imode = 1 To NROOT
                For Ipoin = 1 To Npoin
```

```
                    ShearForcesCQC(Ipoin, 1) += Roij(Imode, Jmode) * ShearForces(Imode, Ipoin,
1) * ShearForces(Jmode, Ipoin, 1)
                    ShearForcesCQC(Ipoin, 2) += Roij(Imode, Jmode) * ShearForces(Imode, Ipoin,
2) * ShearForces(Jmode, Ipoin, 2)
                Next Ipoin
            Next Imode
        Next Jmode

        For Ipoin = 1 To Npoin
            ShearForcesCQC(Ipoin, 1) = Math.Sqrt(ShearForcesCQC(Ipoin, 1))
            ShearForcesCQC(Ipoin, 2) = Math.Sqrt(ShearForcesCQC(Ipoin, 2))
        Next Ipoin

        'and now, finally, calculate storey forces as differences of storey shears
        For Idime = 1 To NDIME
            SeismicForceCQC(Npoin, Idime) = ShearForcesCQC(Npoin, Idime)
        Next Idime
        For Ipoin = Npoin - 1 To 1 Step -1
            For Idime = 1 To NDIME
                SeismicForceCQC(Ipoin,    Idime)    =    ShearForcesCQC(Ipoin,    Idime)    -
ShearForcesCQC(Ipoin + 1, Idime)
            Next Idime
        Next Ipoin

    End Sub
```

6 Calculation Examples

Several examples are presented in this section, comparing MdFem results with the results of Sap4 (in the version of 1994 by Bruce F. Maison, based on the original 1973 Sap4 developed by K. J. Bathe, E. L. Wilson, F. E. Peterson from the University of California, Berkley) and SismiCad (a widly used commercial program by Concrete S.r.L. – Padova – Italy).

All the examples refer to the same spectrum, whose data are listed below.

Number of points = 44

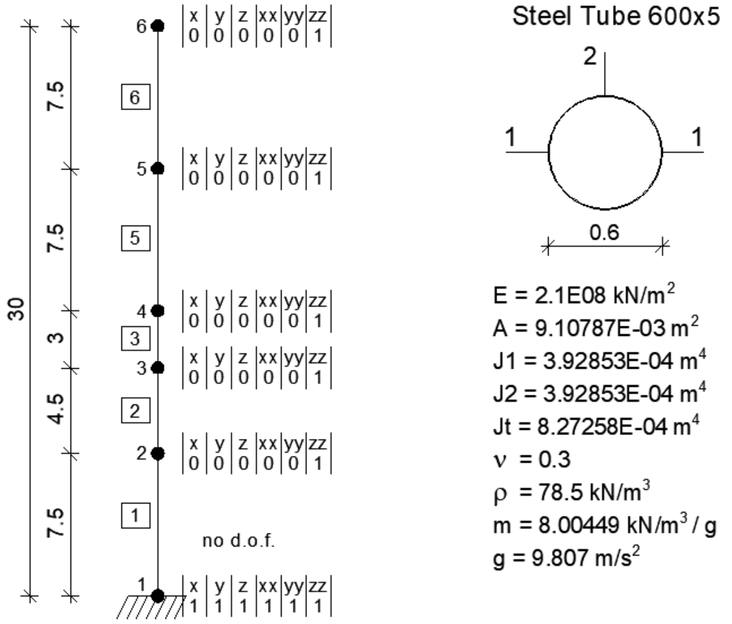
Viscous Damping = 0.05

Behaviour Factor = 1.

period	ag/g	period	ag/g
0.00E+00	1.46E-01	1.87E+00	1.00E-01
2.42E-01	2.57E-01	1.93E+00	9.70E-02
3.63E-01	2.57E-01	2.07E+00	8.39E-02
7.27E-01	2.57E-01	2.22E+00	7.33E-02
7.87E-01	2.38E-01	2.36E+00	6.46E-02
8.47E-01	2.21E-01	2.50E+00	5.74E-02
9.07E-01	2.06E-01	2.65E+00	5.13E-02
9.67E-01	1.93E-01	2.79E+00	4.62E-02
1.03E+00	1.82E-01	2.94E+00	4.18E-02
1.09E+00	1.72E-01	3.08E+00	3.79E-02
1.15E+00	1.63E-01	3.22E+00	3.46E-02
1.21E+00	1.55E-01	3.37E+00	3.17E-02
1.27E+00	1.48E-01	3.51E+00	2.92E-02
1.33E+00	1.41E-01	3.66E+00	2.69E-02
1.39E+00	1.35E-01	3.80E+00	2.49E-02
1.45E+00	1.29E-01	3.94E+00	2.31E-02
1.51E+00	1.24E-01	4.09E+00	2.15E-02
1.57E+00	1.19E-01	4.23E+00	2.01E-02
1.63E+00	1.15E-01	4.38E+00	1.88E-02
1.69E+00	1.11E-01	4.52E+00	1.76E-02
1.75E+00	1.07E-01	4.66E+00	1.65E-02
1.81E+00	1.03E-01	4.81E+00	1.56E-02

6.1 Example 1 – Antennas Pole 1

The structure is an antennas pole, initially without any load, with the characteristics shown in the next figure. The cross section of the steel tube is 600 mm in diameter with a thickness of 5 mm. The restrained degrees of freedom are indicated with the number 1, while the free dof are indicated with the number 1: there are a total number of 25 dof.



Antennas Pole 1 Units: kN, m, s

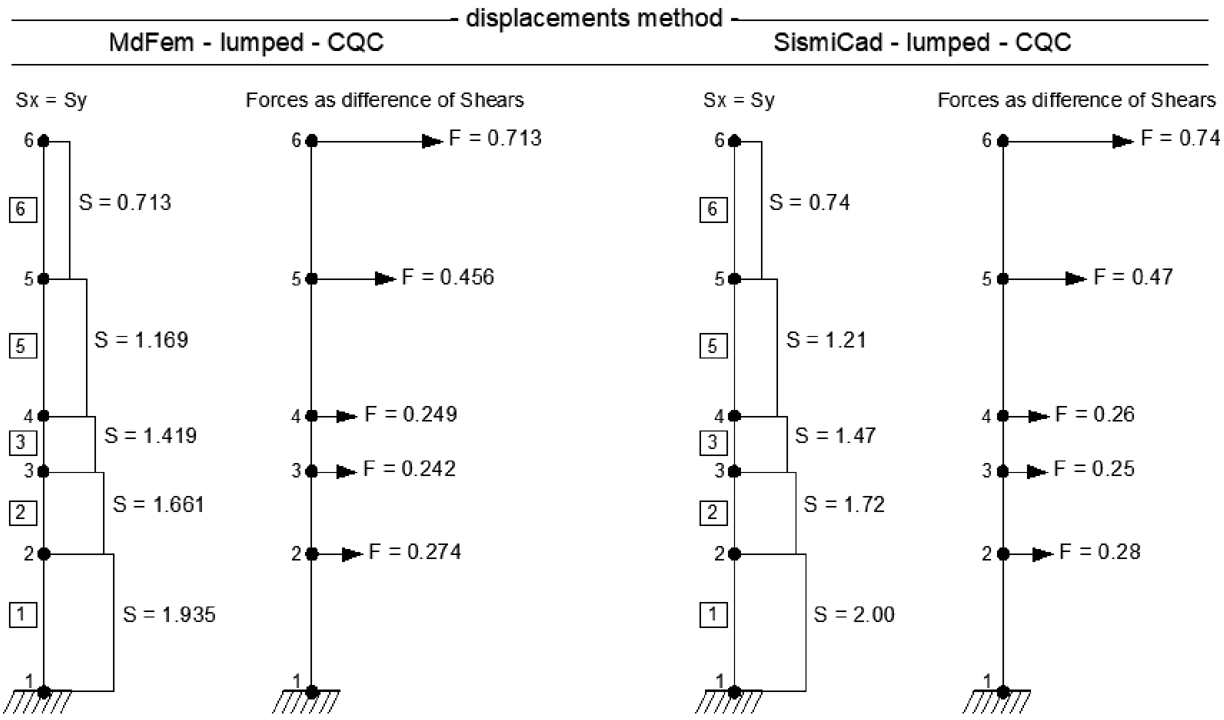
There are symmetries that lead to the same results in couple.

The comparison of the results is shown below.

	Fundamental periods T (s)			participating masses	
Mode	SismiCad	MdFem (lumped)	MdFem (consistent)	x, y	z
1, 2	1.52875	1.5478	1.5118	0.9735 0.9731 0.9433	0 0 0
3, 4	0.2665	0.2689	0.2411		
5, 6	0.10204	0.1024	0.0857		

The difference between the periods calculated with MdFem and SismiCad is approximately 1%, while the difference between the periods calculated with the lumped and the consistent mass matrix increase approximately from 2% (modes 1,2) to 16% (modes 5, 6).

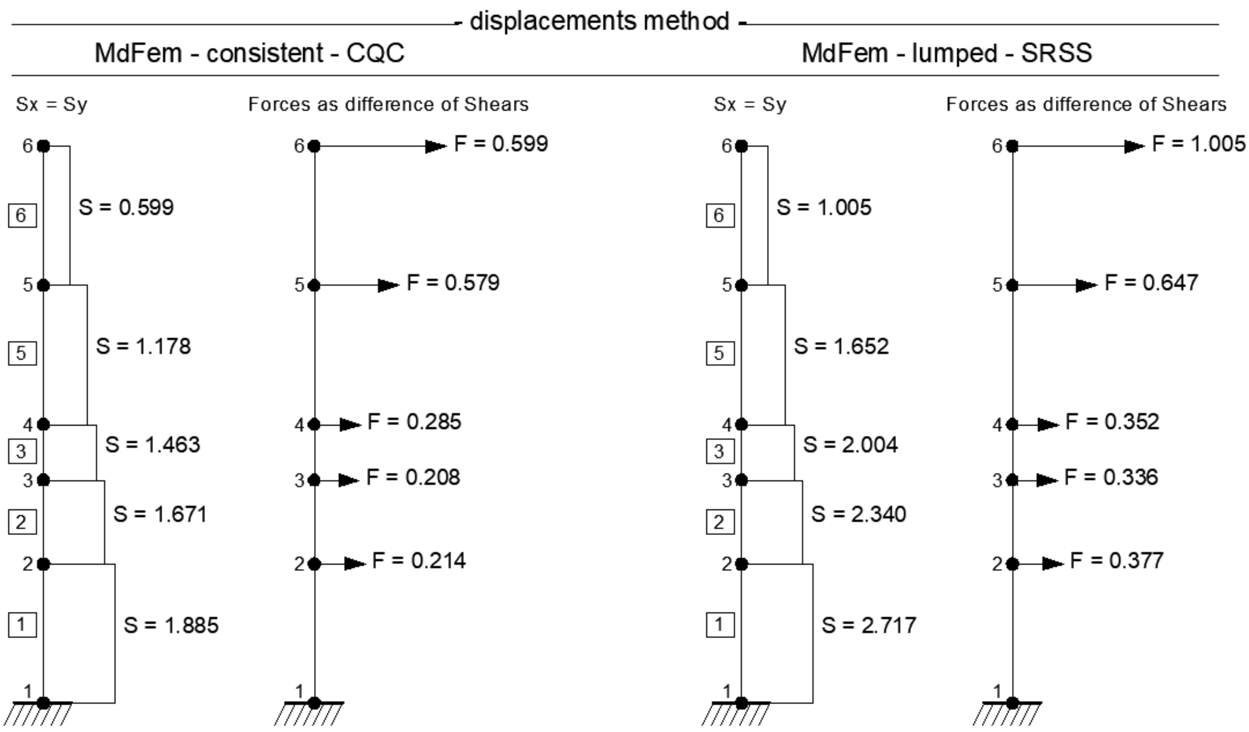
The next figure shows the shear forces calculated with the Response Spectrum Analysis (RSA), by the MdFem and the SismiCad programs, with the displacements method. The forces at the nodes are calculated by hand as the differences of the shear forces, with the aim of comparison with those calculated by MdFem with the forces method. The differences obtained with the two programs are less then 4%.



MdFem – Forces Method
File Antennas Pole 1
combinations of seismic forces

Node	C.Q.C. Force Fx	Fy
1	0.0000E+00	0.0000E+00
2	2.7426E-01	2.7426E-01
3	2.4214E-01	2.4214E-01
4	2.4962E-01	2.4962E-01
5	4.5655E-01	4.5655E-01
6	7.1286E-01	7.1286E-01
base shear	1.9354E+00	1.9354E+00

In the table on the left are listed the seismic forces calculated by the MdFem program with the forces method. In this case the two methods give the same results.



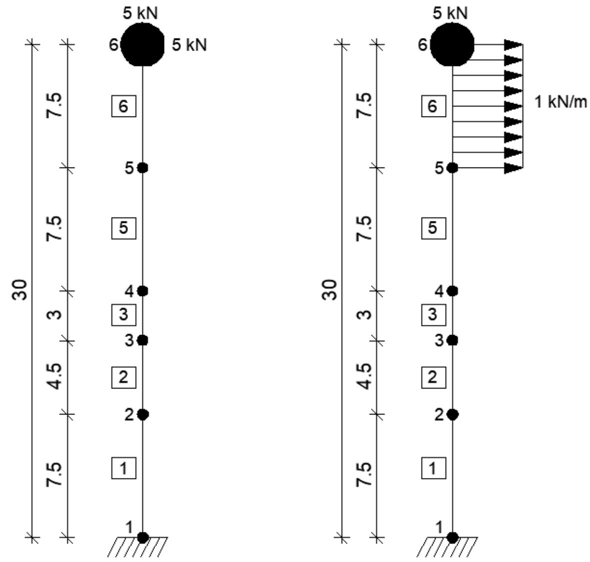
In the previous image, the left side shows the resulting shear forces and nodal forces calculated with the consistent mass matrices and CQC combination, while the right side shows the same values obtained with the lumped mass and SRSS combination. Both the calculations are made by MdFem.

The shear forces obtained with the consistent mass matrices + CQC are slightly lower then those calculated with the lumped mass matrices + CQC (2.6% on base shear).

The shear forces obtained with the lumped mass matrices + SRSS greatly overestimate those obtained with the CQC combination.

Little variations to the described problem are represented in the next figure, where a concentrated load of 5 kN is inserted on the top of the pole (file Antennas Pole 2), and a linear horizontal load is applied to element number 6 (file Antennas Pole 2). <u>In the MdFem program, the loads defined in the load condition number 2 are considered as masses acting in the three dimensions x, y, z</u>.
The results of the first variation are compared with the SismiCad results.

The aim of these new examples is to see how the periods of the natural modes vary with the increase of the masses.



Antennas Pole 2 Antennas Pole 22

Antennas Pole 2

	Fundamental periods T (s)			participating masses	
Mode	SismiCad	MdFem (lumped)	MdFem (consistent)	x, y	z
1, 2	2.09763			0.9783	0
		2.09696		0.9780	0
			2.06923	0.9506	0
3, 4	0.30515	0.30404	0.29140		
5, 6	0.10483	.10742	0.09826		

The results are very close, and the periods have higher values than those without loads.

Antennas Pole 22

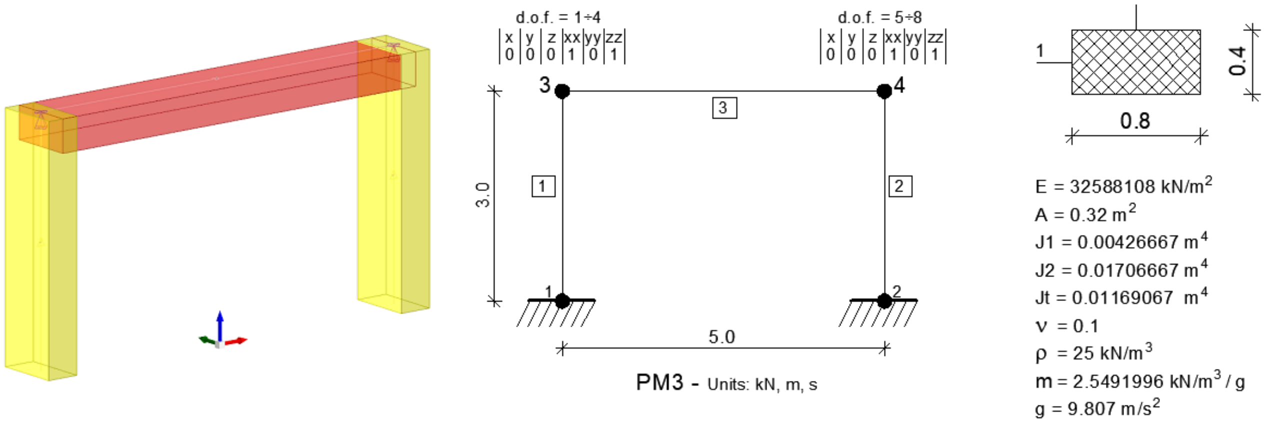
	Fundamental periods T (s)		participating masses	
Mode	MdFem (lumped)	MdFem (consistent)	x, y	z
1, 2	2.56844		0.9814	0
		2.54576	0.9594	0
3, 4	0.33647	0.32606		
5, 6	0.12238	0.11268		

The periods continue to increase, and the periods with consistent mass matrices remain lower. Furthermore, the participant masses with the consistent mass matrices are slightly lower.

6.2 Example 2 – PM3

This example has been already introduced in §3 where a detailed description of the calculation of the participation factors and of the participating masses is shown.

The structure is a single-span, single-storey concrete frame without any load, with the characteristics shown in the next figure. The cross section is 0.8 m x 0.4 m. The restrained degrees of freedom are indicated with the number 1, while the free dof are indicated with the number 0: there are a total number of 8 dof.



The comparison of the results is shown below.

	Fundamental periods T (s)			
Mode	SismiCad	Sap4	MdFem (lumped)	MdFem (consistent)
1	0.05755	0.05002	0.0587	0.0542
2	0.02377	0.02282	0.0228	0.0217
3	0.01945	0.02168	0.0191	0.0182
4	0.00543	0.006087	0.0061	0.0146
5	/	0.006067	0.0061	0.0073
6	/	/	0.0055	0.0042

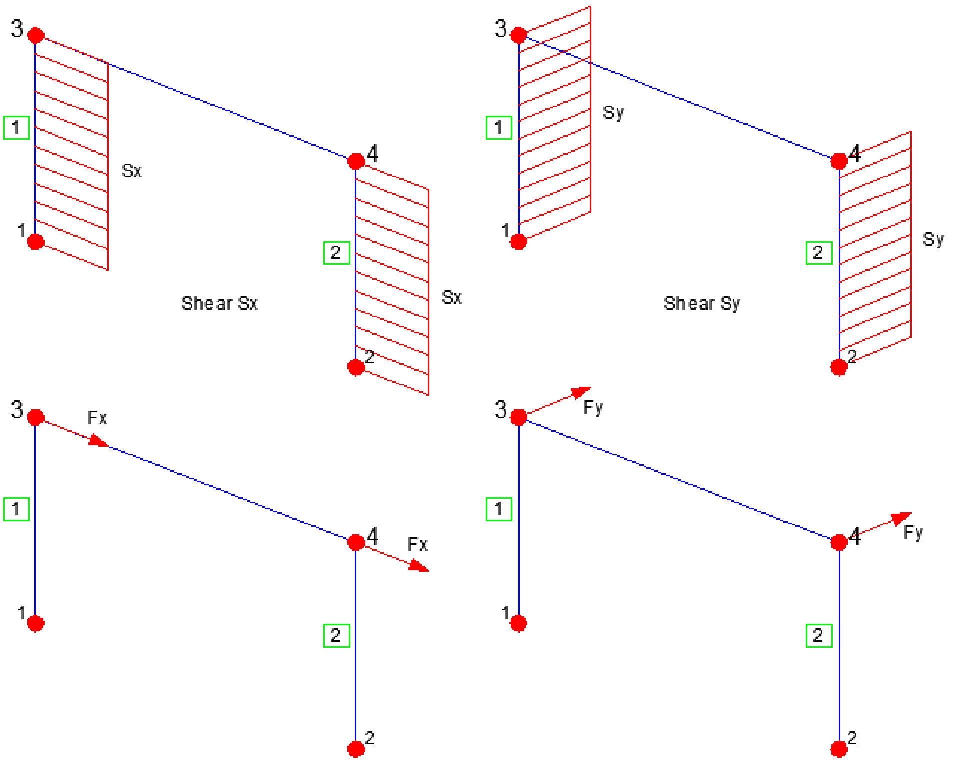
	Modal Participation Factors								
	Sap4			MdFem Lumped			MdFem Consistent		
Mode	x	y	z	x	y	z	x	y	z
1	-2.555	0	0	2.5546	0	0	2.4174	0	0
2	0	-2.555	0	0	2.5546	0	0	2.4283	0
3	0	0	0	0	0	0	0	0	-1.825
4	0	0	-2.555	0	0	2.5546	0	0	0
5	0.0215	0	0	-0.013	0	0	-0.224	0	0
6	/	/	/	0	0	0	0	0	1.221
7	/	/	/	/	/	/	0.051	0	0
8	/	/	/	/	/	/	0	0	0.9437

The calculations with the consistent mass matrices need some more modes, in order to obtain participating masses greater then 0.90 at least in the x, y directions.

	Modal Participating Masses								
	SismiCad			MdFem Lumped			MdFem Consistent		
Mode	x	y	Z	x	y	z	x	y	z
1	1	0	0	0.99999	0	0	0.9035	0	0
2	0	1	0	0	1	0	0	0.9036	0
3	0	0	0	0	0	1	0	0	0.5101
4	0	0	0	0	0	0	0	0	0
5	/	/	/	0.00003	0	0	0.0077	0	0
6	/	/	/	0	0	0	0	0	0.2284
7	/	/	/	/	/	/	0.0004	0	0
8	/	/	/	/	/	/	0	0	0.1365
Sum	1	1	0	1	1	1	0.9035	0.9035	0.875

For a better reading, numbers lower then 1E-10 are written as zeros. The results are in good agreement.

The next figures show the results in terms of shear forces Sx, Sy calculated with the RSA and the displacements method, and the forces at the nodes: in this case of a single storey frame, the forces at the nodes are equal to the shear forces.



Comparisons of RSA results with displacements method

SismiCad lumped		Sap4 lumped				MdFem lumped				MdFem consistent			
CQC		CQC		SRSS		CQC		SRSS		CQC		SRSS	
Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy
5.51	5.02	5.41	5.01	5.41	5.01	5.53	5.01	5.53	5.01	5.01	4.51	5.00	4.51

The results obtained with the different programs are very close. The calculations performed with the consistent mass matrices give lower forces, due to the smaller participating masses.

In this case the shear forces obtained with the SRSS combination are in very good agreement with the CQC results.

Let now control the results obtained with the **forces method**: this method is implemented in MdFem, and the control is made only for the lumped mass matrices calculation.

Also in this case the two methods give the same results. In fact, using two extra decimal digits compared to the table, the base forces calculated with the displacements method are:

Sx = 2 x 5.534 = 11.068

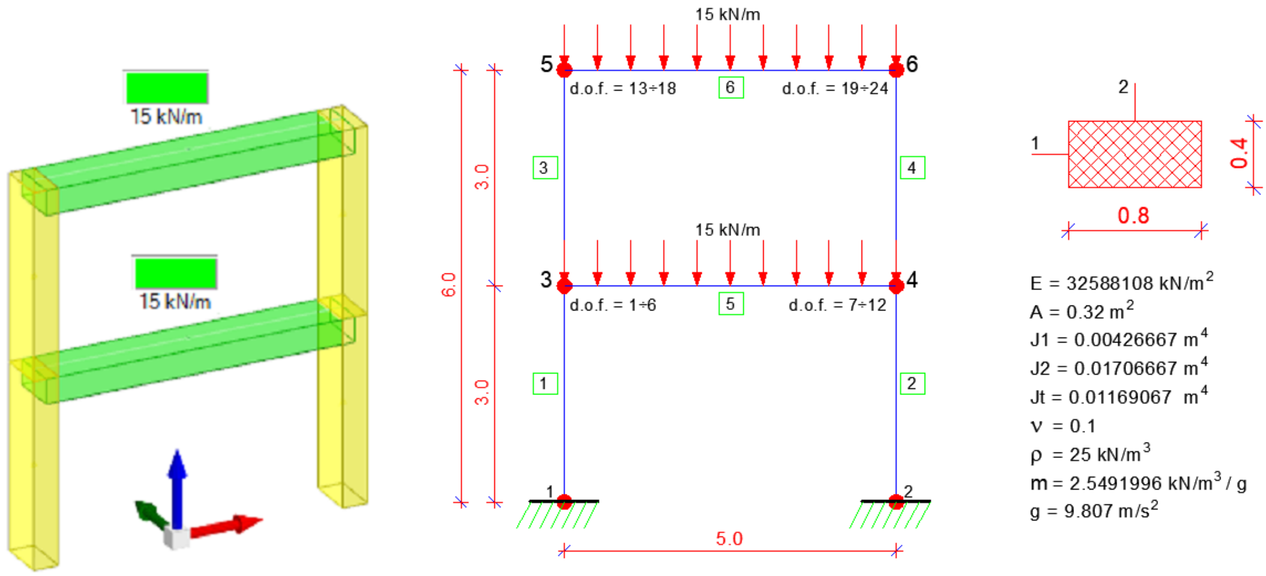
Sy = 2 x 5.007 = 10.014

MdFem – Forces Method
File Antennas Pole 1
combinations of seismic forces

Node	C.Q.C. Force Fx	Fy
1	0.0000E+00	0.0000E+00
2	0.0000E+00	0.0000E+00
3	5.5343E+00	5.0071E+00
4	5.5343E+00	5.0071E+00
base shear	1.1069E+01	1.0014E+01

6.3 Example 3 – Participant Masses 1

The structure is a single-span, two-storeys concrete frame, with the characteristics shown in the next figure. The cross section is 0.8 m x 0.4 m. The horizontal beams have a vertical load of 15 kN/m. The nodes at the base are completely restrained, while the others are completely free: there are a total number of 24 dof.



Participant Masses 1 - Units: kN, m, s

The comparison of the results is shown below.

Mode	Fundamental periods T (s) SismiCad	Sap4	MdFem (lumped)	MdFem (consistent)
1	0.20048	0.2012	0.20122	0.1975
2	0.19895	0.1737	0.17378	0.1721
3	0.16792	0.1311	0.13118	0.1160
4	0.04979	0.0523	0.05231	0.0498
5	0.03258	0.03223	0.03223	0.0318
6	0.03255	0.02993	0.02993	0.0260
7	0.01455	0.01486	0.01486	0.0209
8	/	/	/	0.0160

The results of MdFem and Sap4 are identical, and those of SismiCad are very close.

	Modal Participation Factors								
	Sap4			MdFem Lumped			MdFem Consistent		
Mode	x	y	Z	x	y	z	x	y	z
1	0	-4.896	0	0	4.896	0	0	4.890	0
2	5.196	0	0	-5.196	0	0	5.169	0	0
3	0	0	0	0	0	0	0	0	0
4	1.948	0	0	1.948	0	0	1.842	0	0
5	0	2.611	0	0	2.611	0	0	2.472	0
6	0	0	0	0	0	0	0	0	0
7	0	0	-5.413	0	0	5.413	0	0	2.003
8	/	/	/	/	/	/	0	0	-4.434

For the calculations with the consistent mass matrices, one more mode have been calculated, in order to obtain some more participating masses in the z direction.

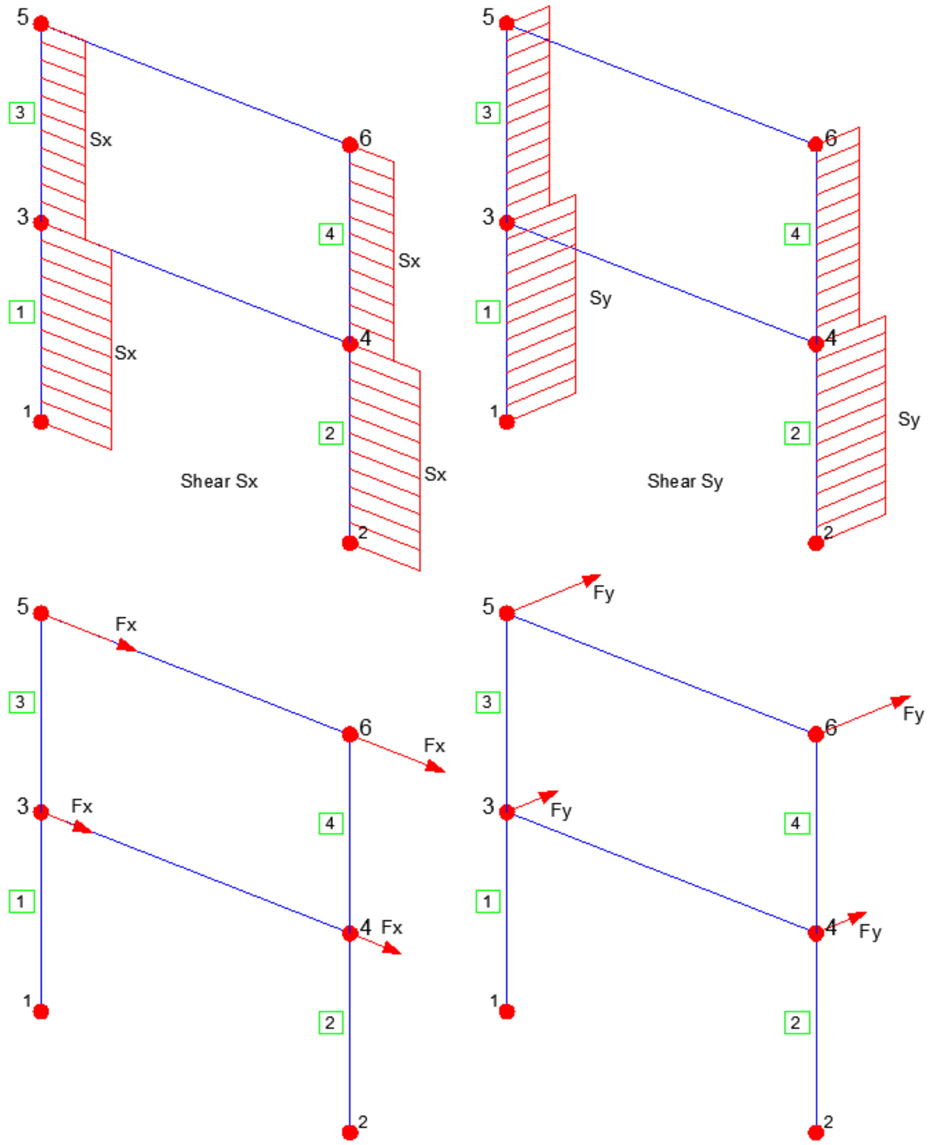
The results of MdFem and Sap4 are identical.

	Modal Participating Masses								
	SismiCad			MdFem Lumped			MdFem Consistent		
Mode	x	y	Z	x	y	Z	x	y	z
1	0	0.7870	0	0	0.7786	0	0	0.7765	0
2	0	0	0	0.8767	0	0	0.8677	0	0
3	0.8852	0	0	0	0	0	0	0	0
4	0.1148	0	0	0.1233	0	0	0.1102	0	0
5	0	0.2130	0	0	0.2214	0	0	0.1984	0
6	0	0	0	0	0	0	0	0	0
7	0	0	0.9544	0	0	0.9514	0	0	0.1303
8	/	/	/	/	/	/	0	0	0.6471
Sum	0.9999	1	0.9544	0.9999	1	0.9514	0.978	0.975	0.777

For a better reading, numbers lower then 1E-10 are written as zeros.

The results are in good agreement: again, with the consistent mass matrices the participating masses are lower.

The next figures show the results in terms of shear forces Sx, Sy calculated with the RSA and the displacements method, and the forces at the nodes calculated by hand as differences of the shear forces.



Comparisons of RSA results with displacements method

	SismiCad lumped		Sap4 lumped				MdFem lumped				MdFem consistent			
	CQC		CQC		SRSS		CQC		SRSS		CQC		SRSS	
Elem.	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy	Sx	Sy
1, 2	30.0	28.8	30.0	28.5	30.0	28.5	30.0	28.5	30.0	28.5	29.6	28.2	29.6	28.1
3, 4	19.3	20.5	19.5	20.5	19.5	20.5	19.5	20.5	19.5	20.5	19.6	19.9	19.0	19.9

The results obtained with the different programs are very close, or identical. With the consistent mass matrices the forces are slightly lower, due to the smaller participating masses.

In this case, again, the shear forces obtained with the SRSS combination are in very good agreement with the CQC results: with the lumped mass matrices the results are the same, while with the consistent mass matrices SRSS values are slightly lower.

Let now control the results obtained with the **forces method**: this method is implemented in MdFem, and the control is made only for the lumped mass matrices calculation.

```
MdFem - Forces Method
File Participant Masses 1L CQC
combinations of seismic forces
---------------------------------------
    Node |       C.Q.C. Force         |
         |     Fx              Fy     |
---------------------------------------
       1    0.0000E+00     0.0000E+00
       2    0.0000E+00     0.0000E+00
       3    1.0907E+01     8.6260E+00
       4    1.0188E+01     7.4505E+00
       5    1.9517E+01     2.0496E+01
       6    1.9517E+01     2.0496E+01
---------------------------------------
base shear  6.0129E+01     5.7069E+01
```

```
MdFem - Displacements Method
Hand calculated forces
From MdFem lumped CQC

---------------------------------
  Node     Fx          Fy
---------------------------------
    1      0.00        0.00
    2      0.00        0.00
    3     10.54        8.04
    4     10.54        8.04
    5     19.52       20.50
    6     19.52       20.50
```

Also in this case, the base shears Sx, Sy, calculated with the forces method, are the same of those calculated with the displacements method: in fact, using an extra decimal digit compared to the table, the base forces calculated with the displacements method are:

Sx = 30.06 x 2 = 60.12

Sy = 28.53 x 2 = 57.06

the two methods give the same results.

The 1^{st} storey forces instead, with the forces method become asymmetric, though their average values are very close to the correct values:

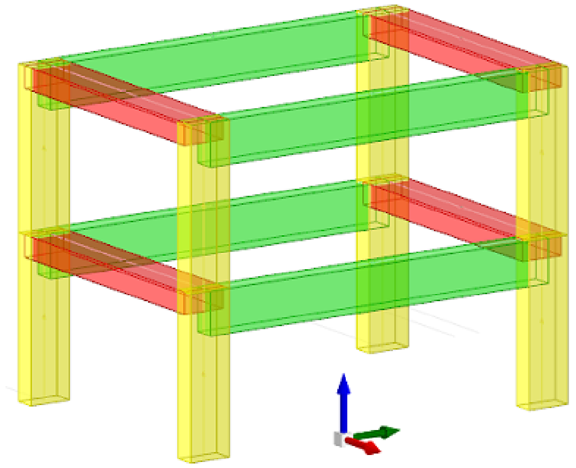
Sx: (10.907 + 10.188) / 2 = 10.55

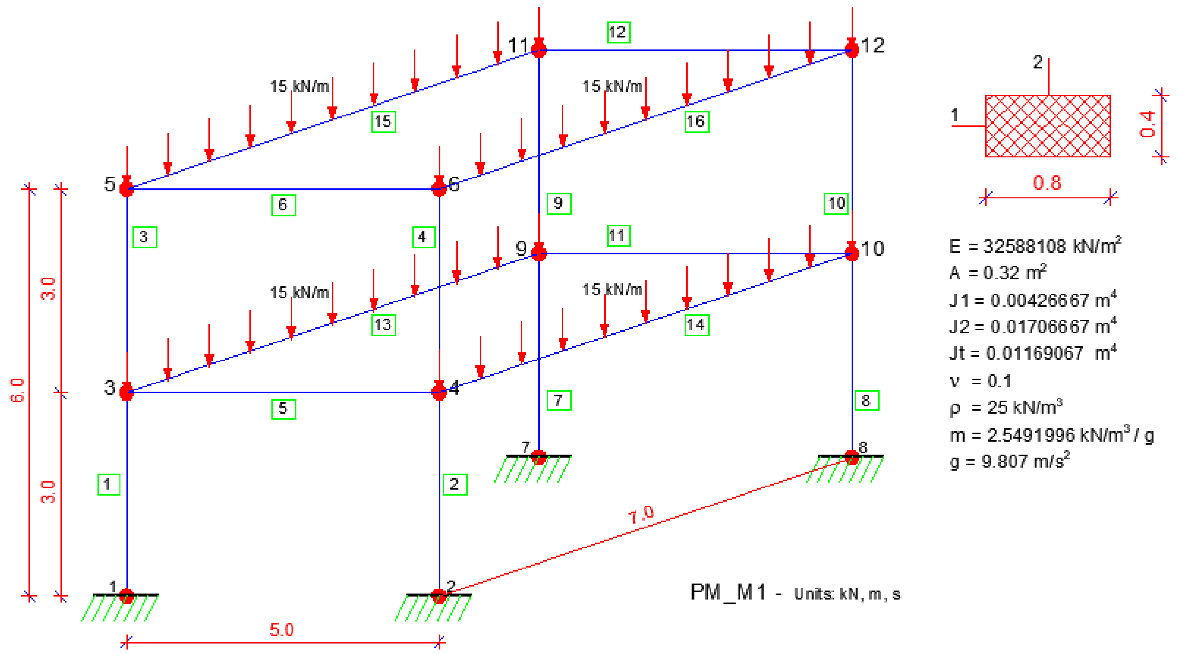
Sy: (8.626 + 7.4505) / 2 = 8.04

As told in 5.4, the implemented method gives good results in some cases, but needs more insights.

6.4 Example 4 – PM_M1

The structure is a two-spans, two-storeys concrete frame, with the characteristics shown in the next figure. The cross section is 0.8 m x 0.4 m and is oriented in different ways as shown in the figure. The horizontal longest beams (with green colour) have a vertical load of 15 kN/m. The nodes at the base are completely restrained, while the others are completely free.





The comparison of the results is shown below.

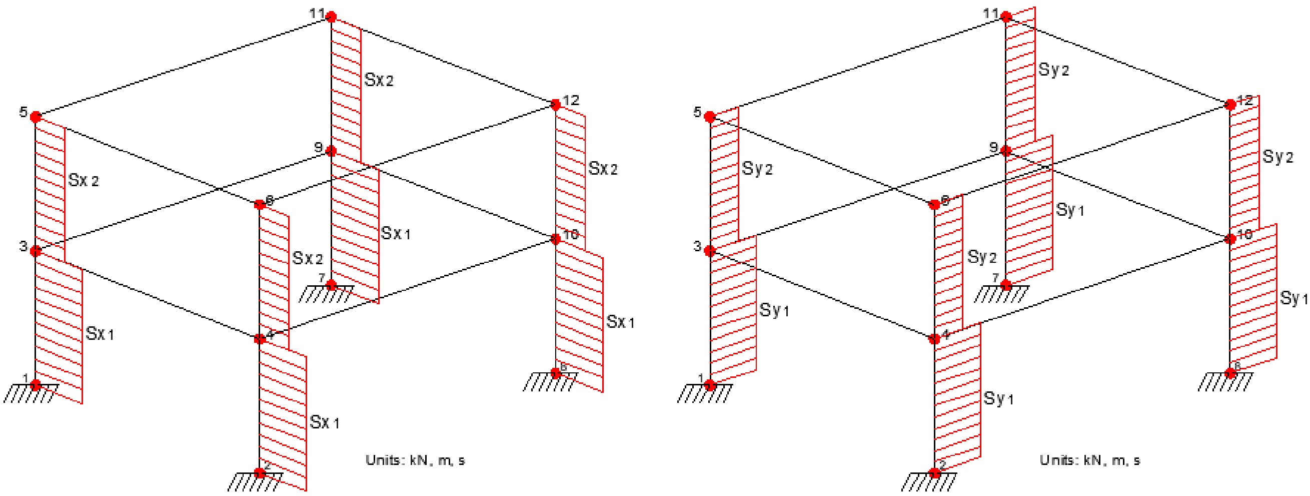
	Fundamental periods T (s)		
Mode	SismiCad	MdFem (lumped)	MdFem (consistent)
1	0.20865	0.21984	0.21863
2	0.17227	0.17067	0.15699
3	0.11612	0.11991	0.11929
4	0.08944	0.09030	0.08535
5	0.05839	0.06502	0.06340
6	0.05660	0.06150	0.05551
7	0.03187	0.03353	0.03298
8	0.03125	0.03204	0.03014
9	0.01794	0.01874	0.02948
10	0.01781	0.01862	0.02516
11	0.01772	0.01853	0.02197
12	0.01760	0.01841	0.01959

The results of MdFem and SismiCad are close.

	Modal Participating Masses								
	SismiCad			MdFem Lumped			MdFem Consistent		
Mode	x	y	Z	x	y	Z	x	y	z
1	0.8880	0	0	0.8772	0	0	0.8714	0	0
2	0	0	0	0	0	0	0	0	0
3	0	0.8800	0	0	0.8640	0	0	0.8580	0
4	0	0	0	0	0	0	0	0	0
5	0.1119	0	0	0.1228	0	0	0.1144	0	0
6	0	0	0	0	0	0	0	0	0
7	0	0.1200	0	0	0.1360	0	0	0.1263	0
8	0	0	0	0	0	0	0	0	0
9	0	0	0.9560	0	0	0.9497	0	0	0
10	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0.4863
12	0	0	0		0	0	00	0	0
Sum	0.9999	0.9999	0.9559	0.9999	0.9999	0.9497	0.9859	0.9844	0.4864

For a better reading, numbers lower then 1E-10 are written as zeros. The results are in good agreement.

The next figure shows the results in terms of shear forces Sx, Sy calculated with the RSA and the displacements method.



Comparisons of RSA results with displacements method

Shear forces	SismiCad lumped	MdFem lumped		MdFem consistent	
	CQC	CQC	SRSS	CQC	SRSS
Sx1	51.10	51.60	51.57	51.12	51.09
Sx2	33.42	34.21	34.23	33.69	33.71
Sy1	41.82	41.51	41.49	41.12	41.10
Sy2	27.65	27.96	27.98	27.55	27.57

The results obtained with the different programs are very close, with minimal differences.

In this example also with the consistent mass matrices the forces are practically the same.

Let now control the results obtained with the **forces method**, comparing again the calculations made with the lumped mass matrices and CQC combination (and MdFem program).

```
MdFem - Forces Method
File Participant Masses 1L CQC
combinations of seismic forces
---------------------------------------
    Node |       C.Q.C. Force          |
         |     Fx              Fy      |
---------------------------------------
       1    0.0000E+00     0.0000E+00
       2    0.0000E+00     0.0000E+00
       3    1.8117E+01     1.4388E+01
       4    1.7731E+01     1.3940E+01
       5    3.3804E+01     2.7500E+01
       6    3.3539E+01     2.7192E+01
       7    0.0000E+00     0.0000E+00
       8    0.0000E+00     0.0000E+00
       9    1.7924E+01     1.4164E+01
      10    1.6853E+01     1.2933E+01
      11    3.4207E+01     2.7961E+01
      12    3.4207E+01     2.7961E+01
---------------------------------------
base shear  2.0638E+02     1.6604E+02
```

```
MdFem - Displacements Method
Hand calculated forces
From MdFem lumped CQC

---------------------------------
  Node     Fx          Fy
---------------------------------
    1      0.00        0.00
    2      0.00        0.00
    3     17.39       13.55
    4     17.39       13.55
    5     34.21       27.96
    6     34.21       27.96
    7      0.00        0.00
    8      0.00        0.00
    9     17.39       13.55
   10     17.39       13.55
   11     34.21       27.96
   12     34.21       27.96
```

In this case again, the base shears Sx, Sy, calculated with the forces method, are the same of those calculated with the displacements method whose values are:

Sx = 51.60 x 4 = 206.4

Sy = 41.51 x 4 = 166.04

Also in this case the forces method leads to asymmetries, though their average values are close to the correct values:

Sx: nodes 3, 4, 9, 10: (18.117 + 17.731 + 17.924 + 16.853) / 4 = 17.656

Sx: nodes 5, 6, 11, 12: (33.804 + 33.539 + 34.207 x 2) / 4 = 33.939

Sy: nodes 3, 4, 9, 10: (14.388 + 13.940 + 14.164 + 12.933) / 4 = 13.856

Sy: nodes 5, 6, 11, 12: (27.50 + 27.192 + 27.961 x 2) / 4 = 27.654

As told in 5.4, the implemented method gives good results in some cases, but needs more insights.

7 Final Remarks

The work presented in this article does not contain anything new from the theoretical point of view, but may be useful for didactical purposes, or even for real applications and research.

The comparisons made with different programs in the various examples, allows me to make some considerations.

The eigenvalues, the eigenvectors and the participating masses have always been calculated with minimal or no differences.

In the Response Spectrum Analysis, the displacements method, with the CQC combination, seems to be the best choice: the results obtained with the different programs are in fact very close and satisfactory. In spite of that, even with this procedure asymmetries may sometimes arise: such type of examples have not been published in this work, but maybe they will be in the future.

The displacements method, with the SRSS combination, gives good results in some cases, but with more exceptions. In the examples here presented it fails, for instance, in the *Antennas Pole 1* case. Actually, it is widely known that it can lead to wrong values, either underestimating or overestimating the correct values [10].

The forces method gives good results in the first two examples, but in the other two some asymmetries arise. Though the presented results are not bad, the implementation described in this article is not as general as needed. As already told, the method needs some insights in order to obtain a procedure of general validity from a mathematical point of view.

One final consideration concerns the use of consistent mass matrices. Though they require an extra computational effort, it is a problem that concerns the computer's CPU and we can ignore it. But they also require the calculation of more natural frequencies to obtain an adequate participating mass, without appreciable improvements in the quality of the results.

8 Bibliography

[1] Bathe K. J.: "*Finite Element Procedures in Engineering Analysis*". Prentice Hall, 1982.

[2] Boreggio G., Benà S.: *"Programma di calcolo SSPACE – Analisi modale di sistemi ad elevato numero di gradi di libertà"*. Lecture notes from University of Padua, 1985.

[3] Bish P., Carvalho E. et alii: *"Eurocode 8: Seismic Design of Buildings - Worked examples"*. Workshop "EC8: Seismic Design of Buildings", Lisbon, 2011.

[4] De Pisapia M.: *"Earthquake – La guida pratica per l'analisi sismica delle strutture".* Private Practice, *2017*.

[5] D.M. 17.01.2018: "*Aggiornamento delle <<Norme tecniche per le costruzioni>>". Supplemento ordinario n° 8 alla GAZZETTA UFFICIALE – Serie generale n° 42.*

[6] Katsikadelis J. T.: *"Dynamic Analysis of Structures"*. Academic Press, Elsevier, 2020.

[7] Eurocode 8 (EN 1998): *"Design of structures for earthquake resistance".*

[8] Serafini P.: *"Semplice esempio numerico del metodo di analisi dinamica". Appunti in rete, 2009.*

[9] Varagnolo P.: *"3-D Beam Finite Element Programming - A Practical Guide: Part 1 – Static Analysis". ResearchGate, 2021.*

[10] Wilson E. L., Der Kiureghian A., Bayo P.: *"A replacement for the SRSS method in seismic analysis". Earthquake Engineering and Structural Dynamics, vol. 9. John Wiley & Sons, 1981.*

[11] Zienkiewicz O. C.: "*The Finite Element Method*". Third Edition. Mc Graw Hill, 1977.

9 Appendix A – Program Sspace

The program execution begins with the subroutine `MainSub()`, which reads the problem data from a file named `InFile`, initializes some variables and then calls the subroutine `Sspace()` where the eigenproblem is solved.

The variables are described and explained inside the subroutine `Sspace()`. For some variables it may be useful to refer to [9].

The order and the type of the data is clear from the context, given the meaning of the variables.

```
Sub MainSub()

    Dim Dummy, Title As String
    Dim File1 As System.IO.StreamWriter = Nothing

    Dim NumFile As Integer = FreeFile()
    FileOpen(NumFile, InFile, OpenMode.Input)

    File1 = My.Computer.FileSystem.OpenTextFileWriter(OuFile, True)

    Title = LineInput(NumFile)
    File1.WriteLine(Title + vbCrLf)

    Dummy = LineInput(NumFile)
    NDOFT = Val(LineInput(NumFile))

    Dummy = LineInput(NumFile)
    NKGLO = Val(LineInput(NumFile))

    Dummy = LineInput(NumFile)
    NMGLO = Val(LineInput(NumFile))

    Dummy = LineInput(NumFile)
    NROOT = Val(LineInput(NumFile))

    ReDim GLOBK(NKGLO), GLOBM(NMGLO), MAXAD(NDOFT + 1)

    Dummy = LineInput(NumFile)
    For Idofn = 1 To NDOFT + 1
        MAXAD(Idofn) = Val(LineInput(NumFile))
    Next Idofn

    Dummy = LineInput(NumFile)
    For i% = 1 To NKGLO
        GLOBK(i%) = Val(LineInput(NumFile))
    Next i%

    Dummy = LineInput(NumFile)
    For i% = 1 To NMGLO
        GLOBM(i%) = Val(LineInput(NumFile))
    Next i%

    NITEM = 16
    IFSS = 1
    IFPR = 0 '1 'flag for complete output (1) or compacted output (0)
    RTOL = 0.000001
```

```
    Call Sspace(File1)

    FileClose()
    File1.Close()

End Sub
```

9.1 Global scope variables

The next table shows the list of global scope variables. Any change of type from single to double or vice versa involves different approximations in the calculations and consequently gives little changes in the results.

```
    Public Errore As Boolean
    Public InFile, OuFile, PrtString, myPath As String
    Public NDOFT, NKGLO, NMGLO, MAXAD() As Integer
    Public GLOBK(), GLOBM() As Double

    Public MassC(21), RTOL, Gaccel As Double
    Public CircFreq(), Frequency(), Period() As Single
    Public NROOT, NITEM, IFSS, IFPR As Integer
    Public Eigenvalues, IfLumped As Boolean
    Public ModFile As String
```

9.2 Other Subroutines

The first subroutine in this paragraph is the `Sspace()` program core, which manages the eigenproblem solution with the calls to the subroutines. The other subroutines follow, in no particular order. The subroutines possibly presented within the text are part of the program and will not be re-presented below.

```
Sub Sspace(File1 As StreamWriter)
    '--------------------------------------------------------------------------------
    ' Program to solve for the smallest eigenvalues and corresponding eigenvectors
    ' in the generalized eigenproblem using the subspace iteration method
    '
    ' written by K. J. Bathe: "Finite Element Procedures in Engineering Analysis"
    '                         Prentice-Hall, 1982
    '
    ' revised and translated in vb.net by Paolo Varagnolo, 2020
    '
    ' Input variables
    '--------------------------------------------------------------------------------
    ' GLOBK(NKGLO)     = stiffness matrix in compacted form
    '                    (global scope variable, already assembled)
    ' GLOBM(NMGLO)     = mass matrix in compacted form
    '                    (global scope variable, already assembled)
    ' MAXAD(NDOFT + 1) = vector containing the addresses of diagonal elements of GLOBK()
    '                    (global scope variable, already assembled)
    ' R(NDOFT, NC)     = eigenvectors on solution exit
    ' EIGV(NC)         = eigenvalues on solution exit
    ' TT(NDOFT)        = working vector
    ' W(NDOFT)         = working vector
    ' AR(NNC)          = working vector storing projection of GLOBK
    ' BR(NNC)          = working vector storing projection of GLOBM
    ' VEC(NC, NC)      = working matrix
    ' D(NC)            = working vector
    ' RTOLV(NC)        = working vector
    ' BUP(NC)          = working vector
    ' BLO(NC)          = working vector
```

```
    ' BUPC(NC)         = working vector
    '
    ' NKGLO            = number of elements below skyline of GLOBK()
    '                    (global scope variable, already assigned)
    ' NMGLO            = number of elements below skyline of GLOBM()
    '                    (global scope variable, already assigned)
    ' NDOFT            = total number of degrees of freedom = order of GLOBK(), GLOBM()
    '                    (global scope variable, already assigned)
    ' NC               = number of iteration vectors used, usually set to MIN(2*NROOT,
NROOT+8),
    '                    cannot be larger then the number of mass degrees of freedom
    ' NNC              = NC * (NC + 1) / 2 dimension of storage vectors AR, BR
    ' NROOT            = number of required eigenvalues and eigenvectors
    '                    (global scope variable, already assigned)
    ' RTOL             = convergence tolerance on eigenvalues (1E-6 or smaller)
    '                    (global scope variable, already assigned)
    ' NITEM            = maximum number of subspace iterations permitted (usually set to 16)
    '                    the parameters NC and/or NITEM must be increased
    '                    if a Then solutions has Not converged
    '                    (global scope variable, already assigned)
    ' NSMAX            = maximum number of sweeps in Jacobi iteration
    ' IFSS             = flag for Sturm sequence check:  = 0 --> no check; = 1 --> check
    '                    (global scope variable, already assigned)
    ' IFPR             = flag for intermediate printing: = 0 --> no check; = 1 --> check
    '                    (global scope variable, already assigned)
    ' Dstif            = scratch streamwriter to store stiffness matrix
    ' File1            = streamwriter for output file
    '-----------------------------------------------------------------------------------------
    ' Output variables
    '-----------------------------------------------------------------------------------------
    ' EIGV(NROOT)      = eigenvalues
    ' R(NDOFT, NROOT)  = eigenvectors
    '-----------------------------------------------------------------------------------------
        Dim NC, MassDOF, NNC, ij, Iconv, NSCH, NSMAX, N1, NC1, ND, ISH, Nite, Nei, Idofn
As Integer
        Dim itemp As Integer
        Dim i%, j%, l%, ii%
        Dim TOLJ, RT, MaxTol, ART, BRT, Dummy, Vnorm, Wnorm, Shift As Double
        Dim Text As String
        Dim ConvReached, Swapped As Boolean

        MassDOF = CalculateMassDOF()
        NC = Math.Min(2 * NROOT, NROOT + 8)
        NC = Math.Min(NC, MassDOF)
        NNC = NC * (NC + 1) / 2

        Dim R(NDOFT, NC), TT(NDOFT), W(NDOFT), EIGV(NC), D(NC), VEC(NC, NC), AR(NNC),
BR(NNC) As Double
        Dim RTOLV(NC), BUP(NC), BLO(NC), BUPC(NC) As Double

        TOLJ = 0.000000000001 'TOLERANCE FOR jACOBI ITERATION

        If NROOT > NDOFT Then
            Text = "The number of requested eigenvalues is greater then " + vbCrLf
            Text += "the number of degrees of freedom in the model." + vbCrLf
            Text += "Only " + NDOFT.ToString + " eigenvalues will be serched." + vbCrLf
            MsgBox(Text, vbExclamation, "Warning")
            File1.WriteLine(Text)
            NROOT = NDOFT
        End If

        If NROOT > MassDOF Then
            Text = "The number of requested eigenvalues is greater then " + vbCrLf
            Text += "the number of mass degrees of freedom." + vbCrLf
            Text += "Only " + MassDOF.ToString + " eigenvalues will be serched." + vbCrLf
            MsgBox(Text, vbExclamation, "Warning")
            File1.WriteLine(Text)
            NROOT = MassDOF
        End If
```

```
        If IFPR <> 0 Then
            Text = vbCrLf
            Text += "global stiffness matrix in compacted form " + vbCrLf
            For i% = 1 To NKGLO
                Text += String.Format("{0,15}", GLOBK(i%).ToString("0.00000000E+00")) +
vbCrLf
            Next i%
            File1.WriteLine(Text)

            Text = vbCrLf
            Text += "global mass matrix in compacted form " + vbCrLf
            For i% = 1 To NMGLO
                Text += String.Format("{0,15}", GLOBM(i%).ToString("0.00000000E+00")) +
vbCrLf
            Next i%
            File1.WriteLine(Text)
        End If

        'Initialization
        Iconv = 0
        NSCH = 0
        NSMAX = 12
        N1 = NC + 1
        NC1 = NC - 1

        Dim Dstif As System.IO.StreamWriter = Nothing
        Dim FileWork0 As String = myPath + "WORK0" 'open a file where will be saved the
stiffness global matrix
        If Dir(FileWork0) <> "" Then Kill(FileWork0)

        Dstif = My.Computer.FileSystem.OpenTextFileWriter(FileWork0, True)

        'write global stiffness matrix
        For i% = 1 To NKGLO
            Dstif.WriteLine(GLOBK(i%))
        Next i%
        Dstif.Close()

        ReDim D(NC), R(NDOFT, NC) 'this set the arrays to zero

        'establish starting iteration vectors
        ND = Int(NDOFT / NC)
        If NMGLO <= NDOFT Then
            j% = 0
            For i% = 1 To NDOFT
                ii% = MAXAD(i%)
                R(i%, 1) = GLOBM(i%)
                If GLOBM(i%) > 0 Then j% += 1
                W(i%) = GLOBM(i%) / GLOBK(ii%)
            Next i%
            If NC > j% Then
                Text = "The number of iteration vectors must not exceed the number of mass
degrees of freedom." + vbCrLf
                MsgBox(Text, vbExclamation, "Warning")
                File1.WriteLine(Text)
                End
            End If
        Else
            For i% = 1 To NDOFT
                ii% = MAXAD(i%)
                R(i%, 1) = GLOBM(ii%)
                W(i%) = GLOBM(ii) / GLOBK(ii)
            Next i%
        End If

        l% = NDOFT - ND
        For j% = 2 To NC
            RT = 0
```

```
            For i% = 1 To l%
                If W(i%) >= RT Then
                    RT = W(i%)
                    ij = i%
                End If
            Next i%
            For i% = l% To NDOFT
                If W(i%) > RT Then
                    RT = W(i%)
                    ij = i%
                End If
            Next i%
            TT(j%) = ij
            W(ij) = 0.0
            l% -= ND
            R(ij, j%) = 1.0
        Next j%

        PrtString = "Degrees of freedom excited by unit starting iteration vectors" +
vbCrLf
        Call PrintArray_1_10_Int(PrtString, TT, 2, NC)
        File1.WriteLine(PrtString)

        'factorize matrix GLOBK() into (L)*(D)*(L(T))
        ISH = 0
        Call Decomp(ISH, File1)
        If Errore Then
            File1.Close() : Exit Sub
        End If

        'start of iteration loop
        Nite = 0
        ConvReached = False
        Do While Iconv = 0 'it is an infinite loop, Iconv remains = 0. The exit from the
loop occurs when ConvReached becomes true
            Nite += 1
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Iteration number: " + String.Format("{0,4}",
Nite.ToString("###0")) + vbCrLf
                File1.WriteLine(Text)
            End If

            'calculate the projection of GLOBK and GLOBM
            ij = 0
            For j% = 1 To NC
                For k% = 1 To NDOFT
                    TT(k%) = R(k%, j%)
                Next k%
                Call REDBAK(TT, File1)
                For i% = j% To NC
                    ART = 0
                    For k% = 1 To NDOFT
                        ART += R(k%, i%) * TT(k%)
                    Next k%
                    ij += 1
                    AR(ij) = ART
                Next i%
                For k% = 1 To NDOFT
                    R(k%, j%) = TT(k%)
                Next k%
            Next j%

            If IFPR <> 0 Then
                PrtString = vbCrLf
                PrtString += "array TT() after REDBAK" + vbCrLf
                Call PrintArray_1_10_Real(PrtString, TT, 1, NDOFT)
                File1.WriteLine(PrtString)
            End If
```

```
        ij = 0
        For j% = 1 To NC
            Call MULT(TT, GLOBM, R, j%, NMGLO)
            For i% = j% To NC
                BRT = 0
                For k% = 1 To NDOFT
                    BRT += R(k%, i%) * TT(k%)
                Next k%
                ij += 1
                BR(ij) = BRT
            Next i%
            If Not ConvReached Then
                For k% = 1 To NDOFT
                    R(k%, j%) = TT(k%)
                Next k%
            End If
        Next j%

        'solve for eigensystem of subspace operators
        If IFPR <> 0 Then
            Call PrintProjections(AR, BR, NC, File1)
        End If

        Call Jacobi(AR, BR, VEC, EIGV, W, NC, NNC, TOLJ, NSMAX, IFPR, File1)
        If Errore Then
            FileClose() : Exit Sub
        End If

        If IFPR <> 0 Then
            Text = "AR and BR after Jacobi diagonalization"
            File1.WriteLine(Text)
            Call PrintProjections(AR, BR, NC, File1)
        End If

        'arrange eigenvalues in ascending order
        Swapped = True
        Do Until Swapped = False
            Swapped = False
            ii = 1
            For i% = 1 To NC1
                itemp = ii + N1 - i%
                If EIGV(i% + 1) < EIGV(i%) Then
                    Swapped = True
                    Dummy = EIGV(i% + 1)
                    EIGV(i% + 1) = EIGV(i%)
                    EIGV(i%) = Dummy
                    Dummy = BR(itemp)
                    BR(itemp) = BR(ii)
                    BR(ii) = Dummy
                    For k% = 1 To NC
                        Dummy = VEC(k%, i% + 1)
                        VEC(k%, i% + 1) = VEC(k%, i%)
                        VEC(k%, i%) = Dummy
                    Next k%
                End If
                ii = itemp
            Next i%
        Loop

        If IFPR <> 0 Then
            Text = "Eigenvalues of AR - Lambda * BR" + vbCrLf
            Call PrintArray_1_10_Real(Text, EIGV, 1, NC)
            File1.WriteLine(Text)
        End If

        'calculate GLOBM times approximate or final eigenvectors
        For i% = 1 To NDOFT
            For j% = 1 To NC
```

```
                    TT(j%) = R(i%, j%)
                Next j%
                For k% = 1 To NC
                    RT = 0
                    For l% = 1 To NC
                        RT += TT(l%) * VEC(l%, k%)
                    Next l%
                    R(i%, k%) = RT
                Next k%
            Next i%

            '---------------------------------------------------------------------------
-----------
            'this is the real exit from the iteration loop
            If ConvReached Then Exit Do
            '---------------------------------------------------------------------------
-----------

            'check for convergence of eigenvalues
            For i% = 1 To NC
                RTOLV(i%) = Math.Abs(EIGV(i%) - D(i%)) / EIGV(i%)
            Next i%
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Relative tolerance reached on eigenvalues." + vbCrLf
                Call PrintArray_1_10_Real(Text, RTOLV, 1, NC)
                File1.WriteLine(Text)
            End If

            MaxTol = -9999
            For i% = 1 To NROOT
                If MaxTol < RTOLV(i%) Then MaxTol = RTOLV(i%)
            Next i%
            If MaxTol < RTOL Then
                'convergence reached
                Text = vbCrLf
                Text += "Convergence reached for tolerance = " + String.Format("{0,12}",
RTOL.ToString("0.00000E+00")) '+ vbCrLf
                File1.WriteLine(Text)
                ConvReached = True 'Iconv = 1
            Else
                If Nite >= NITEM Then
                    'convergence not reached
                    Text = vbCrLf
                    Text += "No convergence in maximum number of iteratioons permitted." +
vbCrLf
                    Text += "Current iteration values will be accepted." + vbCrLf
                    Text += "The Sturm sequence check is not performed." + vbCrLf
                    File1.WriteLine(Text)
                    ConvReached = True 'Iconv = 2
                    IFSS = 0
                Else
                    For i% = 1 To NC
                        D(i%) = EIGV(i%)
                    Next i%
                End If
            End If
        Loop 'end of iteration loop

        Text = vbCrLf
        Text += "The calculated eigenvalues are:" + vbCrLf
        Call PrintArray_1_10_Real(Text, EIGV, 1, NROOT)
        File1.WriteLine(Text)

        Text = ""
        Text += "The calculated eigenvectors are:" + vbCrLf
        For Iroot = 1 To NROOT
            Dim Dumm(NDOFT) As Double
            For Idofn = 1 To NDOFT
```

```
                Dumm(Idofn) = R(Idofn, Iroot)
            Next Idofn
            Call PrintArray_1_10_Real(Text, Dumm, 1, NDOFT)
        Next Iroot
        File1.WriteLine(Text)

        'calculate and print error norms

        'read global stiffness matrix
        Using sr As StreamReader = File.OpenText(FileWork0) 'stiffness global matrix
            For i% = 1 To NKGLO
                GLOBK(i%) = sr.ReadLine
            Next i%
        End Using

        For l% = 1 To NROOT
            RT = EIGV(l%)
            Call MULT(TT, GLOBK, R, l%, NKGLO)
            Vnorm = 0
            For i% = 1 To NDOFT
                Vnorm += TT(i%) * TT(i%)
            Next i%
            Call MULT(W, GLOBM, R, l%, NMGLO)
            Wnorm = 0
            For i% = 1 To NDOFT
                TT(i%) -= RT * W(i%)
                Wnorm += TT(i%) * TT(i%)
            Next i%
            Vnorm = Math.Sqrt(Vnorm)
            Wnorm = Math.Sqrt(Wnorm)
            D(l%) = Wnorm / Vnorm
        Next l%

        If IFPR > 0 Then
            Text = vbCrLf
            Text += "Print error norms on the eigenvalues" + vbCrLf
            Call PrintArray_1_10_Real(Text, D, 1, NROOT)
            File1.WriteLine(Text)
        End If

        'apply Sturm sequence check
        If IFSS <> 0 Then 'IFSS is the flag for Sturm sequence check
            Call SturmCheck(EIGV, RTOLV, BUP, BLO, BUPC, D, NC, Nei, RTOL, Shift, File1)
            If Errore Then
                File1.Close() : Exit Sub
            End If
            If IFPR > 0 Then
                Text = vbCrLf
                Text += "Check applied at shift: " + String.Format("{0,12}",
Shift.ToString("0.00000E+00")) '+ vbCrLf
                File1.WriteLine(Text)
            End If

            'shift matrix GLOBK
            'read global stiffness matrix
            Using sr As StreamReader = File.OpenText(FileWork0) 'stiffness global matrix
file
                For i% = 1 To NKGLO
                    GLOBK(i%) = sr.ReadLine
                Next i%
            End Using

            If NMGLO <= NDOFT Then
                For i% = 1 To NDOFT
                    ii = MAXAD(i%)
                    GLOBK(ii) -= GLOBM(i%) * Shift
                Next i%
            Else
                For i% = 1 To NKGLO
```

```
                    GLOBK(i%) -= GLOBM(i%) * Shift
                Next i%
            End If

            'factorize shifted matrix
            ISH = 1
            Call Decomp(ISH, File1)
            If Errore Then
                File1.Close() : Exit Sub
            End If

            'count number of negative diagonal elements
            NSCH = 0
            For i% = 1 To NDOFT
                ii = MAXAD(i%)
                If GLOBK(ii) < 0 Then NSCH += 1
            Next i%

            If NSCH = Nei Then
                Text = ""
                Text += "We found the lowest: " + String.Format("{0,4}",
NSCH.ToString("###0")) + " eigenvalues "
                Text += "(" + NROOT.ToString + " had to be found)" + vbCrLf
                File1.WriteLine(Text)
            Else
                Text = ""
                Text += "There are: " + String.Format("{0,4}", (NSCH -
Nei).ToString("###0")) + " eigenvalues missing" + vbCrLf
                File1.WriteLine(Text)
            End If
        End If 'Sturm sequence check

        'Write FREQUENCIES (ADDED BY PV)
        If IFSS = 0 Then
            Text = vbCrLf + " PRINT OF FREQUENCIES" + vbCrLf
        Else
            Text = " PRINT OF FREQUENCIES" + vbCrLf
        End If
        Text += vbCrLf
        Text += " MODE          CIRCULAR " + vbCrLf
        Text += " NUMBER        FREQUENCY        FREQUENCY        PERIOD" + vbCrLf
        Text += "              (RAD/SEC)       (CYCLES/SEC)       (SEC)" + vbCrLf
        Text += "--------------------------------------------------------------"
        File1.WriteLine(Text)

        ReDim CircFreq(NROOT), Frequency(NROOT), Period(NROOT)
        Dim TPI As Double = 8 * Math.Atan(1)
        For i% = 1 To NROOT
            CircFreq(i%) = Math.Sqrt(EIGV(i%)) 'circular frequency
            Frequency(i%) = CircFreq(i%) / TPI 'frequency
            Period(i%) = TPI / CircFreq(i%) 'period
            Text = String.Format("{0,5}", i%.ToString("####0")) + "    "
            Text += String.Format("{0,17}", CircFreq(i%).ToString("E8"))
            Text += String.Format("{0,17}", Frequency(i%).ToString("E8"))
            Text += String.Format("{0,17}", Period(i%).ToString("E8"))
            File1.WriteLine(Text)
        Next i%



    Sub Decomp(ISH As Integer, File1 As StreamWriter)

        'subroutine to factorize stiffness matrix GLOBK() into (L)*(D)*(L(T))

        'In the original subroutine GLOBK, MAXAD, NDOFT are formal arguments, but here
these variables have a global scope

        Dim KN, KL, KU, KH, IC, KLT, KI, ND, KK As Integer
        Dim k%
```

```
        Dim Var1, Var2 As Double
        Dim Text As String

        If NDOFT = 1 Then Return

        For n% = 1 To NDOFT
            KN = MAXAD(n%)
            KL = KN + 1
            KU = MAXAD(n% + 1) - 1
            KH = KU - KL
            If KH > 0 Then
                k% = n% - KH
                IC = 0
                KLT = KU
                For j% = 1 To KH
                    IC += 1
                    KLT -= 1
                    KI = MAXAD(k%)
                    ND = MAXAD(k% + 1) - KI - 1
                    If ND > 0 Then
                        KK = Math.Min(IC, ND)
                        Var1 = 0
                        For l% = 1 To KK
                            Var1 += GLOBK(KI + l%) * GLOBK(KLT + l%)
                        Next l%
                        GLOBK(KLT) -= Var1
                    End If
                    k% += 1
                Next j%
            End If
            If KH >= 0 Then
                k% = n%
                Var2 = 0
                For KK = KL To KU
                    k% -= 1
                    KI = MAXAD(k%)
                    Var1 = GLOBK(KK) / GLOBK(KI)
                    If Math.Abs(Var1) > 10000000.0 Then
                        Text = vbCrLf
                        Text += "Stop - Sturm sequence check failed because of multiplier
growth" + vbCrLf
                        Text += "for column number " + n%.ToString + ". Multiplier = " +
String.Format("{0,12}", Var1.ToString("0.00000E+00"))
                        MsgBox(Text, vbExclamation, "Warning")
                        File1.WriteLine(Text)
                        Errore = True : Exit Sub
                    End If
                    Var2 += Var1 * GLOBK(KK)
                    GLOBK(KK) = Var1
                Next KK
                GLOBK(KN) -= Var2
            End If

            If GLOBK(KN) <= 0 Then
                If ISH = 0 Then
                    Text = vbCrLf
                    Text += "Stop - Stiffness matrix not positive definite." + vbCrLf
                    Text += "non positive pivot for equation " + n%.ToString + ". Pivot =
" + String.Format("{0,12}", GLOBK(KN).ToString("0.00000E+00"))
                    MsgBox(Text, vbExclamation, "Warning")
                    File1.WriteLine(Text)
                    Errore = True : Exit Sub
                Else
                    If GLOBK(KN) = 0 Then GLOBK(KN) = -0.0000000000000001
                End If
            End If
        Next n%

    End Sub
```

```
    Sub REDBAK(ByRef Vett() As Double, File1 As StreamWriter)

        'subroutine to reduce and back-substitute iteration vectors

        'In the original subroutine GLOBK, MAXAD, NDOFT are formal arguments, but here
these variables have a global scope

        Dim KL, KU As Integer
        Dim k%, n%
        Dim Var1 As Double

        For n% = 1 To NDOFT
            KL = MAXAD(n%) + 1
            KU = MAXAD(n% + 1) - 1
            If (KU - KL) >= 0 Then
                k% = n%
                Var1 = 0
                For kk = KL To KU
                    k% -= 1
                    Var1 += GLOBK(kk) * Vett(k%)
                Next kk
                Vett(n%) -= Var1
            End If
        Next n%

        For n% = 1 To NDOFT
            k% = MAXAD(n%)
            Vett(n%) = Vett(n%) / GLOBK(k%)
        Next n%

        If NDOFT = 1 Then Return

        n% = NDOFT
        For l% = 2 To NDOFT
            KL = MAXAD(n%) + 1
            KU = MAXAD(n% + 1) - 1
            If (KU - KL) >= 0 Then
                k% = n%
                For kk = KL To KU
                    k% -= 1
                    Vett(k%) -= GLOBK(kk) * Vett(n%)
                Next kk
            End If
            n% -= 1
        Next l%

    End Sub
```

```
    Sub MULT(ByRef TT() As Double, Vett() As Double, RR(,) As Double, Icolu As Integer,
MaxInd As Integer)

        'subroutine to evaluate product of Vett() times the Icolu-th column of RR(,) and
store result in TT()
        'Vett() can be the global Stiffness Matrix or the global Mass Matrix
        'MaxInd is the number of elements in the Stiffness Matrix (NKGLO) or Mass Matrix
(NMGLO)

        'In the original subroutine MAXAD, NDOFT are formal arguments, but here these
variables have a global scope

        Dim KL, KU, II As Integer
        Dim Var1, Var2 As Double
```

```
        ReDim TT(NDOFT)

        If MaxInd = NDOFT Then
            'lumped mass matrix
            For i% = 1 To NDOFT
                TT(i%) = Vett(i%) * RR(i%, Icolu)
            Next i%
            Return
        End If

        'consistent mass matrix or stiffnes matrix
        For i% = 1 To NDOFT
            KL = MAXAD(i%)
            KU = MAXAD(i% + 1) - 1
            II = i% + 1
            Var1 = RR(i%, Icolu)
            For kk = KL To KU
                II -= 1
                TT(II) += Vett(kk) * Var1
            Next kk
        Next i%

        If NDOFT = 1 Then Return

        For i% = 2 To NDOFT
            KL = MAXAD(i%) + 1
            KU = MAXAD(i% + 1) - 1
            If KU - KL >= 0 Then
                II = i%
                Var2 = 0
                For kk = KL To KU
                    II -= 1
                    Var2 += Vett(kk) * RR(II, Icolu)
                Next kk
                TT(i%) += Var2
            End If
        Next i%

    End Sub
```

```
    Sub PrintProjections(AR() As Double, BR() As Double, NC As Integer, File1 As
StreamWriter)
        'print projections of stiffness and mass matrix

        Dim Text As String
        Dim Itemp, N1 As Integer
        Dim ii%

        N1 = NC + 1

        Text = vbCrLf
        Text += "Projection of stiffness matrix" + vbCrLf
        ii% = 1
        For i% = 1 To NC
            Itemp = ii% + NC - i%
            Call PrintArray_1_10_Real(Text, AR, ii%, Itemp)
            ii% += N1 - i%
        Next i%
        Text += "Projection of mass matrix" + vbCrLf
        ii% = 1
        For i% = 1 To NC
            Itemp = ii% + NC - i%
            Call PrintArray_1_10_Real(Text, BR, ii%, Itemp)
            ii% += N1 - i%
        Next i%
        File1.WriteLine(Text)
```

```
    End Sub
```

```
    Sub Jacobi(ByRef A() As Double, ByRef B() As Double, ByRef X(,) As Double, ByRef
EIGV() As Double, ByRef D() As Double, NC As Integer,
               NWA As Integer, TOLJ As Double, NSMAX As Integer, IFPR As Integer, File1 As
StreamWriter)

        'solve the generalized eigenproblem using the generalized Jacobi iteration

        Dim Text As String
        Dim n1, ii, Nsweep, nr, jp1, jm1, ljk, jj, kp1, km1, jk, kk, im1, ij, ik, lji, lki
As Integer
        Dim ji, ki, lkj As Integer
        Dim Eps, EptolA, EptolB, AKK, AJJ, AB, Check, SQCH, D1, D2, DEN, CA, CG As Double
        Dim AJ, BJ, AK, BK, xj, xk, Tol, Dif, EpsA, EpsB, BB, Dummy As Double
        Dim ConvReached As Boolean

        n1 = NC + 1
        ii = 1
        For i% = 1 To NC
            If A(ii) <= 0 Or B(ii) <= 0 Then
                Text = vbCrLf
                Text += "Error: solution stop" + vbCrLf
                Text += "matrices not positive definite" + vbCrLf
                Text += "Index: " + ii.ToString + vbCrLf
                Text += "Stiffness = " + String.Format("{0,12}",
A(ii).ToString("0.0000E+00")) + vbCrLf
                Text += "Mass      = " + String.Format("{0,12}",
B(ii).ToString("0.0000E+00")) + vbCrLf
                File1.WriteLine(Text)
                Errore = True : Exit Sub
            End If

            D(i%) = A(ii) / B(ii)
            EIGV(i%) = D(i%)
            ii += n1 - i%
        Next i%

        ReDim X(NC, NC)
        For i% = 1 To NC
            X(i%, i%) = 1
        Next i%

        If NC = 1 Then Exit Sub

        'initialize sweep counter and begin iteration
        Nsweep = 0
        nr = NC - 1
        Do While Nsweep < NSMAX
            Nsweep += 1
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Jacobi subroutine: sweep no. " + Nsweep.ToString + vbCrLf
                File1.WriteLine(Text)
            End If

            'check if present off-diagonal element is large enough to require zeroing
            Eps = (0.01 ^ Nsweep) ^ 2
            For j% = 1 To nr
                jp1 = j% + 1
                jm1 = j% - 1
                ljk = jm1 * NC - Int(jm1 * j% / 2)
                jj = ljk + j%
                For k% = jp1 To NC
                    kp1 = k% + 1
```

```
                     km1 = k% - 1
                     jk = ljk + k%
                     kk = km1 * NC - Int(km1 * k% / 2) + k%
                     EptolA = (A(jk) * A(jk)) / (A(jj) * A(kk))
                     EptolB = (B(jk) * B(jk)) / (B(jj) * B(kk))
                     If EptolA >= Eps Or EptolB >= Eps Then
                         'if zeroing is required, calculate the rotation matrix elements CA
and CG
                         AKK = A(kk) * B(jk) - B(kk) * A(jk)
                         AJJ = A(jj) * B(jk) - B(jj) * A(jk)
                         AB = A(jj) * B(kk) - A(kk) * B(jj)
                         Check = (AB * AB + 4 * AKK * AJJ) / 4
                         If Check < 0 Then
                             Text = vbCrLf
                             Text += "Error: solution stop" + vbCrLf
                             Text += "matrices not positive definite" + vbCrLf
                             File1.WriteLine(Text)
                             Errore = True : Exit Sub
                         End If
                         SQCH = Math.Sqrt(Check)
                         D1 = AB / 2 + SQCH
                         D2 = AB / 2 - SQCH
                         DEN = D1
                         If Math.Abs(D2) > Math.Abs(D1) Then DEN = D2
                         If DEN = 0 Then
                             CA = 0
                             CG = -A(jk) / A(kk)
                         Else
                             CA = AKK / DEN
                             CG = -AJJ / DEN
                         End If

                         'perform the generalized rotation, to zero the present off-
diagonal element
                         If NC - 2 <> 0 Then
                             If jm1 - 1 >= 0 Then
                                 For i% = 1 To jm1
                                     im1 = i% - 1
                                     ij = im1 * NC - Int(im1 * i% / 2) + j%
                                     ik = im1 * NC - Int(im1 * i% / 2) + k%
                                     AJ = A(ij)
                                     BJ = B(ij)
                                     AK = A(ik)
                                     BK = B(ik)
                                     A(ij) = AJ + CG * AK
                                     B(ij) = BJ + CG * BK
                                     A(ik) = AK + CA * AJ
                                     B(ik) = BK + CA * BJ
                                 Next i%
                             End If
                             If kp1 - NC <= 0 Then
                                 lji = jm1 * NC - Int(jm1 * j% / 2)
                                 lki = km1 * NC - Int(km1 * k% / 2)
                                 For i% = kp1 To NC
                                     ji = lji + i%
                                     ki = lki + i%
                                     AJ = A(ji)
                                     BJ = B(ji)
                                     AK = A(ki)
                                     BK = B(ki)
                                     A(ji) = AJ + CG * AK
                                     B(ji) = BJ + CG * BK
                                     A(ki) = AK + CA * AJ
                                     B(ki) = BK + CA * BJ
                                 Next i%
                             End If
                             If jp1 - km1 <= 0 Then
                                 lji = jm1 * NC - Int(jm1 * j% / 2)
                                 For i% = jp1 To km1
```

```
                                    ji = lji + i%
                                    im1 = i% - 1
                                    ik = im1 * NC - Int(im1 * i% / 2) + k%
                                    AJ = A(ji)
                                    BJ = B(ji)
                                    AK = A(ik)
                                    BK = B(ik)
                                    A(ji) = AJ + CG * AK
                                    B(ji) = BJ + CG * BK
                                    A(ik) = AK + CA * AJ
                                    B(ik) = BK + CA * BJ
                                Next i%
                            End If
                        End If
                        AK = A(kk)
                        BK = B(kk)
                        A(kk) = AK + 2 * CA * A(jk) + CA * CA * A(jj)
                        B(kk) = BK + 2 * CA * B(jk) + CA * CA * B(jj)
                        A(jj) += 2 * CG * A(jk) + CG * CG * AK
                        B(jj) += 2 * CG * B(jk) + CG * CG * BK
                        A(jk) = 0
                        B(jk) = 0

                        'update the eigenvector matrix after each rotation
                        For i% = 1 To NC
                            xj = X(i%, j%)
                            xk = X(i%, k%)
                            X(i%, j%) = xj + CG * xk
                            X(i%, k%) = xk + CA * xj
                        Next i%
                    End If 'If EptolA >= Eps Or EptolB >= Eps
                Next k%
            Next j%

            'update the eigenvalues after each sweep
            ii = 1
            For i% = 1 To NC
                If A(ii) <= 0 Or B(ii) <= 0 Then
                    Text = vbCrLf
                    Text += "Error: solution stop" + vbCrLf
                    Text += "matrices not positive definite" + vbCrLf
                    Text += "Index: " + ii.ToString + vbCrLf
                    Text += "Stiffness = " + String.Format("{0,12}",
A(ii).ToString("0.0000E+00")) + vbCrLf
                    Text += "Mass      = " + String.Format("{0,12}",
B(ii).ToString("0.0000E+00")) + vbCrLf
                    File1.WriteLine(Text)
                    Errore = True : Exit Sub
                End If
                EIGV(i%) = A(ii) / B(ii)
                ii += n1 - i%
            Next i%
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Current eigenvalues in Jacobi are:" + vbCrLf
                Call PrintArray_1_10_Real(Text, EIGV, 1, NC)
                File1.WriteLine(Text)
            End If

            'check for convergence
            Dim EigenOK As Boolean = True
            For i% = 1 To NC
                Tol = TOLJ * D(i%)
                Dif = Math.Abs(EIGV(i%) - D(i%))
                If Dif > Tol Then
                    EigenOK = False
                    Exit For
                End If
            Next i%
```

```
            ConvReached = True
            Eps = TOLJ ^ 2
            EpsA = 1.0E-20
            EpsB = 1.0E-20
            If EigenOK Then
                'check all off-diagonal elements to see if another sweep is required
                EpsA = 1.0E-50
                EpsB = 1.0E-50
                For j% = 1 To nr
                    jm1 = j% - 1
                    jp1 = j% + 1
                    lkj = jm1 * NC - Int(jm1 * j% / 2)
                    jj = lkj + j%
                    For k% = jp1 To NC
                        km1 = k% - 1
                        jk = lkj + k%
                        kk = km1 * NC - Int(km1 * k% / 2) + k%
                        Dummy = (A(jk) * A(jk)) / (A(jj) * A(kk))
                        If EpsA < Dummy Then EpsA = Dummy
                        Dummy = (B(jk) * B(jk)) / (B(jj) * B(kk))
                        If EpsB < Dummy Then EpsB = Dummy
                    Next k%
                Next j%
            Else
                '280 stuff
            End If

            If EpsA > Eps Or EpsB > Eps Then
                ConvReached = False
            End If

            'update D matrix and start a new sweep, if allowed
            '280 stuff: an extra D matrix update can be made, but I admit this for a
better readability of the program
            For i% = 1 To NC
                D(i%) = EIGV(i%)
            Next i%

            If ConvReached Then
                'fill out bottom triangle of resultant metrices and scale eigenvectors
                ii = 1
                For i% = 1 To NC
                    BB = Math.Sqrt(B(ii))
                    For k% = 1 To NC
                        X(k%, i%) = X(k%, i%) / BB
                    Next k%
                    ii += n1 - i%
                Next i%
                Exit Do
            End If
        Loop

    End Sub
```

```
    Sub SturmCheck(EIGV() As Double, RTOLV() As Double, BUP() As Double, BLO() As Double,
BUPC() As Double, NEIV() As Double,
                   NC As Integer, ByRef nei As Integer, RTOL As Double, ByRef Shift As
Double, File1 As StreamWriter)

        'subroutine to evaluate shift for Sturm sequence check

        Dim NROOTtmp, LM, ll As Integer
        Dim l%, i%
        Dim Ftol As Double
        Dim Text As String
```

```
        Ftol = 0.01
        For i% = 1 To NC
            BUP(i%) = EIGV(i%) * (1 + Ftol)
            BLO(i%) = EIGV(i%) * (1 - Ftol)
        Next i%

        NROOTtmp = 0
        For i% = 1 To NC
            If RTOLV(i%) < RTOL Then NROOTtmp += 1
        Next i%
        If NROOTtmp < 1 Then
            Text = vbCrLf
            Text += "Error: solution stop in SturmCheck" + vbCrLf
            Text += "no eigenvalues found." + vbCrLf
            File1.WriteLine(Text)
            Errore = True : Exit Sub
        End If

        If NROOTtmp < NROOT Then NROOT = NROOTtmp

        'find upper bounds on eigenvalues clusters
        For i% = 1 To NROOTtmp
            NEIV(i%) = 1
        Next i%

        If NROOTtmp = 1 Then
            BUPC(1) = BUP(1)
            LM = 1
            l% = 1
            i% = 2
        Else
            l% = 1
            i% = 2
            Do While i% <= NROOTtmp
                If BUP(i% - 1) > BLO(i%) Then
                    NEIV(l%) += 1
                    i += 1
                Else
                    BUPC(l%) = BUP(i% - 1)
                    If i% <= NROOTtmp Then
                        l% += 1
                        i% += 1
                        If i% > NROOTtmp Then BUPC(l%) = BUP(i% - 1)
                    End If
                End If
            Loop
            LM = l%
        End If

        If NROOTtmp <> NC Then
            Do While NROOTtmp < NC
                If BUP(i% - 1) <= BLO(i%) Then Exit Do
                If RTOLV(i%) >= RTOL Then Exit Do
                BUPC(l%) = BUP(i%)
                NEIV(l%) += 1
                NROOTtmp += 1
                i% += 1
            Loop
        End If

        'find shift
        If IFPR > 0 Then
            Text = vbCrLf
            Text += "Upper bounds of eigenvalue clusters" + vbCrLf
            Call PrintArray_1_10_Real(Text, BUPC, 1, LM)
            File1.WriteLine(Text)
            Text = vbCrLf
            Text += "Number of eigenvalues in each cluster" + vbCrLf
```

```
            Call PrintArray_1_10_Int(Text, NEIV, 1, LM)
            File1.WriteLine(Text)
        End If
        ll = LM - 1
        If LM > 1 Then
            Do Until l% = 1
                For i% = 1 To ll
                    NEIV(l%) += NEIV(i%)
                Next i%
                l% -= 1
                ll -= 1
            Loop
        End If

        If IFPR > 0 Then
            Text = vbCrLf
            Text += "Number of eigenvalues less then upper bounds" + vbCrLf
            Call PrintArray_1_10_Int(Text, NEIV, 1, LM)
            File1.WriteLine(Text)
        End If

        l% = 0
        i% = 0
        Do While i% < LM
            i% += 1
            l% += 1
            If NEIV(i%) >= NROOTtmp Then Exit Do
        Loop
        Shift = BUPC(l%)
        nei = NEIV(l%)

    End Sub
```

```
    Function CalculateMassDOF()
        'calculate the number of mass degrees of freedom
        'mass matrix GLOBM and MAXAD have a global scope

        Dim Index As Integer

        CalculateMassDOF = 0

        If NMGLO = NDOFT Then
            'lumped mass matrix
            For Idofn = 1 To NDOFT
                If GLOBM(Idofn) <> 0 Then CalculateMassDOF += 1
            Next Idofn
        Else
            'consistent mass matrix
            For Idofn = 1 To NDOFT
                Index = MAXAD(Idofn)
                If GLOBM(Index) <> 0 Then CalculateMassDOF += 1
            Next Idofn
        End If

    End Function
```

```
    Sub PrintArray_1_10_Int(ByRef Text As String, Goofy() As Double, First As Integer,
Last As Integer)
        'add to Text the 1 dimensional array Goofy, in clusters of 10 integer numbers in
every row
        'starting from First element, to Last element

        Dim Ntens, i1, i2 As Integer
```

```
        Ntens = Int((Last - First) / 10)
        i2 = First - 1

        If Ntens > 0 Then
            For Itens = 1 To Ntens
                i1 = 10 * (Itens - 1) + First
                i2 = i1 + 9
                For Ielem = i1 To i2 - 1
                    Text += String.Format("{0,5}", Goofy(Ielem).ToString("####0"))
                Next Ielem
                Text += String.Format("{0,5}", Goofy(i2).ToString("####0")) + vbCrLf
            Next Itens
        End If

        'print last elements
        Dim Nelem = (Last - First) + 1 - 10 * Ntens

        i1 = i2 + 1
        i2 = i1 + Nelem - 1

        If Nelem > 0 Then
            For Ielem = i1 To i2 - 1
                Text += String.Format("{0,5}", Goofy(Ielem).ToString("####0"))
            Next Ielem
            Text += String.Format("{0,5}", Goofy(i2).ToString("####0")) '+ vbCrLf
        End If

    End Sub
```

```
    Sub PrintArray_1_10_Real(ByRef Text As String, Goofy() As Double, First As Integer,
Last As Integer)
        'add to Text the 1 dimensional array Goofy, in clusters of 10 real numbers in
every row
        'starting from First element, to Last element

        Dim Ntens, i1, i2 As Integer

        Ntens = Int((Last - First) / 10)
        i2 = First - 1

        If Ntens > 0 Then
            For Itens = 1 To Ntens
                i1 = 10 * (Itens - 1) + First
                i2 = i1 + 9
                For Ielem = i1 To i2 - 1
                    Text += String.Format("{0,12}", Goofy(Ielem).ToString("0.0000E+00"))
                Next Ielem
                Text += String.Format("{0,12}", Goofy(i2).ToString("0.0000E+00")) + vbCrLf
            Next Itens
        End If

        'print last elements
        Dim Nelem = (Last - First) + 1 - 10 * Ntens

        i1 = i2 + 1
        i2 = i1 + Nelem - 1

        If Nelem > 0 Then
            For Ielem = i1 To i2 - 1
                Text += String.Format("{0,12}", Goofy(Ielem).ToString("0.0000E+00"))
            Next Ielem
            Text += String.Format("{0,12}", Goofy(i2).ToString("0.0000E+00")) + vbCrLf
        End If

    End Sub
```

10 Appendix B – Program MdFem

The routines that have not yet been presented are listed hereunder. Some missing routines may be found in [9].

The program execution begins with the subroutine **`MDfem_Execute`**, which initializes the variables, reads the problem data, calculates the elements' local axes and finally calls the subroutine **`SubSpace`** where the dynamic analysis is performed.

The subroutine that reads the problem data will not be presented, for the aim of this work is to focus on the substance of FEM programming. The program can nevertheless be used, with the data directly inserted in a specifically re-written subroutine **`ReadMdFemFile`**, presented in §10.1: the data are those of example **`Participant Masses 1`** in §6.3 (with lumped mass matrix and CQC combination). Data input is usually performed in a graphic pre-processor environment, that records all the values in a file, respecting precise specifications: describing and listing these subroutines would have required a discussion that is beyond the scope of this work.

```
Sub MDfem_Execute(dtmStart As Date)

    Dim x1, x2, y1, y2, z1, z2 As Double

    FileClose()

    Call InitVariables()

    Call ReadMdFemFile()

    'calculate transformation T matrix for all the elements: Tmat(,,)
    For Ielem = 1 To Nelem
        x1 = Xcoor(Inc1(Ielem)) : y1 = Ycoor(Inc1(Ielem)) : z1 = Zcoor(Inc1(Ielem))
        x2 = Xcoor(Inc2(Ielem)) : y2 = Ycoor(Inc2(Ielem)) : z2 = Zcoor(Inc2(Ielem))

        Call LocalAxes_Vitaliani_PV(Ielem, x1, y1, z1, x2, y2, z2)
    Next Ielem

    If Eigenvalues Then
        Call SubSpace() 'calculate vibration modes
    End If
    Call MDFEM(dtmStart) 'calculate FEM static analysis

End Sub
```

10.1 Input data

```
Sub ReadMdFemFile()

    'this is not the real reading routine:
    'it only sets the data for the tutorial "Participant Masses 1L CQC" example

    Dim Icase, Ielem, Ipoin As Integer

    RTOL = 0.000001
    NITEM = 16
    IFSS = 1
    IFPR = 0

    Tit1$ = "Participating Masses 1L CQC - units: kN, m, s"
    Npoin = 6
    Nelem = 6
    Ncase = 2
    Ntype = 2

    IFPOS = 1
    Eigenvalues = True
    NROOT = 7
    IfLumped = True
    NITEM = 16
    IFSS = 1
    IFPR = 0
    Gaccel = 9.807

    ' ..................................................
    Nmats = 2
    NCOMB = 0
    Ngaps = 0

    Call MdFemArrayDimensions()

    'nodal coordinates
    Xcoor = {0, 0, 5, 0, 5, 0, 5}
    Ycoor = {0, 0, 0, 0, 0, 0, 0}
    Zcoor = {0, 0, 0, 3, 3, 6, 6}

    'fixity codes
    Iffix(1) = 1 : Iffix(2) = 1
    Iffiy(1) = 1 : Iffiy(2) = 1
    Iffiz(1) = 1 : Iffiz(2) = 1
    Ifrxx(1) = 1 : Ifrxx(2) = 1
    Ifryy(1) = 1 : Ifryy(2) = 1
    Ifrzz(1) = 1 : Ifrzz(2) = 1

    For Ipoin = 1 To Npoin
        IDDOF(1, Ipoin) = Iffix(Ipoin)
        IDDOF(2, Ipoin) = Iffiy(Ipoin)
        IDDOF(3, Ipoin) = Iffiz(Ipoin)
        IDDOF(4, Ipoin) = Ifrxx(Ipoin)
        IDDOF(5, Ipoin) = Ifryy(Ipoin)
        IDDOF(6, Ipoin) = Ifrzz(Ipoin)
    Next Ipoin

    'properties
    PROPS(1, 1) = 32588108.0 'Young modulus
    PROPS(1, 2) = 0.32       'Area
    PROPS(1, 3) = 0.01706667 'Jx
    PROPS(1, 4) = 0.00426667 'Jy
    PROPS(1, 5) = 0          'Width (only for Winkler elements)
    PROPS(1, 6) = 25.0       'Weight density
    PROPS(1, 7) = 2.5491996  'Mass density
    PROPS(1, 8) = 0.01169067 'torsional inertia
    PROPS(1, 9) = 0.1        'Poisson ratio
```

```
PROPS(1, 10) = 0.5 * PROPS(1, 1) / (1 + PROPS(1, 9)) 'G modulus
TrazFlag(1) = 0          'not an only tension element

PROPS(Nmats, 1) = PROPS(1, 1)
PROPS(Nmats, 2) = PROPS(1, 2)
PROPS(Nmats, 3) = 0.00426667 'Jx
PROPS(Nmats, 4) = 0.01706667 'Jy
PROPS(Nmats, 5) = PROPS(1, 5)
PROPS(Nmats, 6) = PROPS(1, 6)
PROPS(Nmats, 7) = PROPS(1, 7)
PROPS(Nmats, 8) = PROPS(1, 8)
PROPS(Nmats, 9) = PROPS(1, 9)
PROPS(Nmats, 10) = PROPS(1, 10)
TrazFlag(Nmats) = TrazFlag(1)

'element definition
NelOnlyTraz = 0
Mater = {0, 1, 1, 1, 1, 2, 2}
Inc1 = {0, 1, 2, 3, 4, 3, 5}
Inc2 = {0, 3, 4, 5, 6, 4, 6}

'loads data
Icase = 1
Titl$(Icase) = "C=1 - G1: own weights"
Gravity_Case(Icase) = -3

Icase = 2
Titl$(Icase) = "C=2 - G2: permanent LOADS"
For Ielem = 5 To 6
    LoadType(Icase, Ielem) = 1
    Dload(Icase, Ielem, 1) = 0     'x load at node i
    Dload(Icase, Ielem, 2) = 0     'y load at node i
    Dload(Icase, Ielem, 3) = -15.0 'z load at node i
    Dload(Icase, Ielem, 4) = Dload(Icase, Ielem, 1) 'x load at node j
    Dload(Icase, Ielem, 5) = Dload(Icase, Ielem, 2) 'y load at node j
    Dload(Icase, Ielem, 6) = Dload(Icase, Ielem, 3) 'z load at node j
Next Ielem

'Spectrum data
NpoiSpec = 44
DampCsi = 0.05
BehFact = 1.0
IfCQC = True
Ifspe = 1
ReDim Spectrum(NpoiSpec, 2)

Spectrum = {
    {0.0, 0.0, 0.0},
    {0.0, 0.00E+00, 0.146},
    {0.0, 0.242, 0.257},
    {0.0, 0.363, 0.257},
    {0.0, 0.727, 0.257},
    {0.0, 0.787, 0.238},
    {0.0, 0.847, 0.221},
    {0.0, 0.907, 0.206},
    {0.0, 0.967, 0.193},
    {0.0, 1.03, 0.182},
    {0.0, 1.09, 0.172},
    {0.0, 1.15, 0.163},
    {0.0, 1.21, 0.155},
    {0.0, 1.27, 0.148},
    {0.0, 1.33, 0.141},
    {0.0, 1.39, 0.135},
    {0.0, 1.45, 0.129},
    {0.0, 1.51, 0.124},
    {0.0, 1.57, 0.119},
    {0.0, 1.63, 0.115},
    {0.0, 1.69, 0.111},
    {0.0, 1.75, 0.107},
    {0.0, 1.81, 0.103},
    {0.0, 1.87, 0.1},
```

```
            {0.0, 1.93, 0.097},
            {0.0, 2.07, 0.0839},
            {0.0, 2.22, 0.0733},
            {0.0, 2.36, 0.0646},
            {0.0, 2.5, 0.0574},
            {0.0, 2.65, 0.0513},
            {0.0, 2.79, 0.0462},
            {0.0, 2.94, 0.0418},
            {0.0, 3.08, 0.0379},
            {0.0, 3.22, 0.0346},
            {0.0, 3.37, 0.0317},
            {0.0, 3.51, 0.0292},
            {0.0, 3.66, 0.0269},
            {0.0, 3.8, 0.0249},
            {0.0, 3.94, 0.0231},
            {0.0, 4.09, 0.0215},
            {0.0, 4.23, 0.0201},
            {0.0, 4.38, 0.0188},
            {0.0, 4.52, 0.0176},
            {0.0, 4.66, 0.0165},
            {0.0, 4.81, 0.0156}
            }

    End Sub
```

10.2 Global scope variables

The next table shows the list of global scope variables.

The global scope variables listed in the MdFem program published in [9] must be added to these.

Any change of type from single to double or vice versa involves different approximations in the calculations and consequently gives little changes in the results.

```
    Public dtmStart, dtmEnd As Date

    Public GLOBM(), MassC(21), RTOL, Gaccel As Double
    Public CircFreq(), Frequency(), Period() As Double
    Public NMGLO, NROOT, NITEM, IFSS, IFPR As Integer
    Public Eigenvalues, IfLumped As Boolean
    Public ModFile As String
    Public PartFact(,), PartMas(,), TotalMass() As Double
    Public NodalMass(), ModalShape(,,), SpecDisp(,,), CombStruDisp() As Double
    Public Roij(,) As Double
    Public ShearForcesCQC(Npoin, NDIME), ShearForcesSRSS(Npoin, NDIME) As Double
    Public SeismicForceCQC(,), SeismicForceSRSS(,) As Double
    Public ValAg_g, ValF0, ValTc, DampCsi, S_s, C_c, S_t, ModalAcc() As Double
    Public IfCQC As Boolean
    Public SpeFile As String
    Public Ifspe, NpoiSpec As Integer
    Public BehFact, Spectrum(,) As Single
```

10.3 Inizializations and array dimensioning

```
    Sub MdFemArrayDimensions()

        'main arrays dimensioning

        ReDim Iffix(Npoin + Ngaps), Iffiy(Npoin + Ngaps), Iffiz(Npoin + Ngaps), Ifrxx(Npoin
 + Ngaps), Ifryy(Npoin + Ngaps), Ifrzz(Npoin + Ngaps)
        ReDim Xcoor(Npoin + Ngaps), Ycoor(Npoin + Ngaps), Zcoor(Npoin + Ngaps)
        ReDim SPRIN(NDOFN, Npoin + Ngaps), GAPS(3, Npoin + Ngaps)
        ReDim Inc1(Nelem + Ngaps), Inc2(Nelem + Ngaps)
```

```
        ReDim Mater(Nelem + Ngaps), OnlyTraz(Nelem + Ngaps)
        ReDim IDDOF(NEVAB, Npoin + Ngaps)
        ReDim XYCOO(NEVAB, Nelem + Ngaps), IJINC(Nelem + Ngaps, 2)
        ReDim LMDOF(NEVAB, Nelem + Ngaps)

        ReDim Comb_Factor(NCOMB, Ncase), Tit_Comb(NCOMB), GravityAmplif(NCOMB)

        ReDim TrazFlag(Nmats), PROPS(Nmats + Ngaps + 1, 11)

        ReDim GlobF(Ncase + NCOMB + NROOT, Nelem + Ngaps, 2 * NDOFN), TREAC(Ncase + NCOMB
+ NROOT, Npoin + Ngaps, NDOFN)
        ReDim Displ(Ncase + NCOMB + NROOT, Npoin, NDOFN)
        ReDim Stre1(Ncase + NCOMB + NROOT, Nelem + Ngaps, NSTRE + 2), Stre2(Ncase + NCOMB
+ NROOT, Nelem + Ngaps, NSTRE + 2)

        ReDim Titl$(Ncase + NCOMB + NROOT), Dload(Ncase + NCOMB + NROOT, Nelem + Ngaps,
6)
        ReDim Gravity_Case(Ncase + NCOMB + NROOT), PointLoad(Ncase + NCOMB + NROOT, Npoin
+ Ngaps, NDOFN)
        ReDim LoadType(Ncase + NCOMB + NROOT, Nelem + Ngaps), NpointLoads(Ncase + NCOMB +
NROOT), NspanLoads(Ncase + NCOMB + NROOT)
        ReDim Rx3D(Ncase + NCOMB), Ry3D(Ncase + NCOMB), Rz3D(Ncase + NCOMB), Rxx3D(Ncase
+ NCOMB), Ryy3D(Ncase + NCOMB), Rzz3D(Ncase + NCOMB)

        ReDim Tmat(Nelem, 6, 6) 'Tmat is the transformation matrix from local to global
coordinates

    End Sub
```

10.4 Other Subroutines

The first subroutine in this paragraph is the **SubSpace** program core, which manages the dynamic analysis with the calls to the main subroutines. The other subroutines follow, in no particular order. The subroutines presented within the text are part of the program and will not be re-presented below.

```
    Sub SubSpace()
        ' ======================================================================
        ' this subroutine comes from Sub MDFEM and is modified
        ' in order to obtain the smallest eigenvalues and the corresponding eigenvectors,
        ' according to the SSPACE program published in K. J. Bathe: "Finite Element Procedures
In Engineering Analysis
        '
        '            WRITTEN BY:  P. VARAGNOLO                    == pvi20 ==
        ' ======================================================================

        Dim FileDummy As String
        Dim FileWork1 As String = myPath + "WORK1" 'open a file where will be saved the
element GLOBAL stiffness matrices
        Dim FileWork3 As String = myPath + "WORK3" 'open a file where will be saved the
element LOCAL  stiffness matrices
        Dim File1 As System.IO.StreamWriter = Nothing
        Dim File2 As System.IO.StreamWriter = Nothing 'a dummy file to be killed after it's
use

        Errore = False

        If Ngaps > 0 Then
            Dim Text = "Non linear elements such as Gaps not allowed for Modal Analysis."
            MsgBox(Text, vbExclamation, "Warning")
            Exit Sub
        End If

        If NelOnlyTraz > 0 Then
            Dim Text = "Non linear elements such as tie-beams not allowed for Modal Analysis."
            MsgBox(Text, vbExclamation, "Warning")
```

```
            Exit Sub
        End If

        If Dir(ModFile, 0) <> "" Then Kill(ModFile)
        FileDummy = DataFile.Substring(0, Len(DataFile) - 4) + ".sel" : If Dir(FileDummy, 0) 
<> "" Then Kill(FileDummy)
        If Dir(myPath + "work1", 0) <> "" Then Kill(myPath + "work1")        'kill work1 if 
it exists from previous calculations
        If Dir(myPath + "work3", 0) <> "" Then Kill(myPath + "work3")        'kill work3 if 
it exists from previous calculations

        File1 = My.Computer.FileSystem.OpenTextFileWriter(ModFile, True)
        File2 = My.Computer.FileSystem.OpenTextFileWriter(FileDummy, True)

        Call Write1_Eigen(File1)

        Call SetNodalData(File2) 'File2 is the Dummy streamwriter
        If Errore Then
            FileClose() : Exit Sub
        End If

        Call Write2_Eigen(File1) 'WriteProperties

        ' *** NUMERATION OF DEGREES OF FREEDOM
        Call DOFNUM()

        ' *** CALL ELEMENT SUBROUTINE (+ generate gaps elements)
        Call INPELE(File2)  'File2 is the Dummy streamwriter

        ' *** CALCULATE ADDRESS OF DIAGONAL ELEMENTS
        Call ADDRES()

        ' *** DEAL WITH LOAD CASES AND LOAD COMBINATIONS (the combinations are treated as 
Load Cases)

        ReDim DloaL(Ncase + NCOMB + NROOT, Nelem + Ngaps, 2 * NDIME) 'carichi distribuiti in 
coordinate locali

        For Icase = 1 To Ncase + NCOMB
            Call LOADS(Icase, 1, File2) 'File2 is the Dummy streamwriter
        Next Icase

        Call CreateStiffnessMatrix(FileWork1, FileWork3)   ' *** CALCULATE ELASTIC STIFFNESS 
MATRIX
        Call CreateMassMatrix()                             ' *** CALCULATE MASS MATRIX

        Call Sspace(File1) ' *** calculate eigenvalues and eigenvectors
        If Errore Then
            FileClose()
            Dim Text = "Error calculating eigenvalues."
            MsgBox(Text, vbExclamation, "Warning")
        End If

        If Ifspe > 0 Then
            Call STRBER(0, 1, Nelem, FileWork3)
            Call OutSTRBE(0, 1, Nelem, File1, File2)

            Call SeismicForcesCalculationCQC()
            Call WriteSeismicForces(File1)
        End If

        'restore original IDDOF(,) values - those read from input data
        For Ipoin = 1 To Npoin
            If Ntype = 2 Or Ntype = 3 Then
                IDDOF(1, Ipoin) = Iffix(Ipoin)
                IDDOF(2, Ipoin) = Iffiy(Ipoin)
                IDDOF(3, Ipoin) = Iffiz(Ipoin)
                IDDOF(4, Ipoin) = Ifrxx(Ipoin)
                IDDOF(5, Ipoin) = Ifryy(Ipoin)
                IDDOF(6, Ipoin) = Ifrzz(Ipoin)
            End If
```

```
        Next Ipoin

        FileClose()
        File1.Close()
        File2.Close() : Kill(FileDummy)        'this dynamic module made use of some of the
static routines: now the static output file must be killed
        If Dir(myPath + "WORK1", 0) <> "" Then Kill(myPath + "WORK1")
        If Dir(myPath + "WORK3", 0) <> "" Then Kill(myPath + "WORK3")

    End Sub
```

```
    Sub CreateMassMatrix()

        Dim Icase As Integer
        ReDim TotalMass(NDOFN), NodalMass(Npoin)

        Call AssembMasses() ' *** CALL MASS SUBROUTINE

        ' *** ASSEMBLE LOADS MASSES
        If Ncase > 1 Then
            Icase = 2 'dead non structural loads
            Call AddLoadMass(Icase)
        End If

    End Sub
```

```
    Sub AssembMasses()

        'CREATE ELEMENT MASS MATRICES AND ASSEMBLE GLOBAL MASS MATRIX

        Dim Ielem As Integer

        If IfLumped Then
            NMGLO = NDOFT
        Else
            NMGLO = NKGLO
        End If

        ReDim GLOBM(NMGLO)

        ' *** LOOP OVER ELEMENT GROUPS

        For Ielem = 1 To Nelem
            If Ntype = 2 Or Ntype = 3 Then Call BeamMass(Ielem)
        Next Ielem

    End Sub
```

```
    Sub BeamMass(Ielem As Integer)

        'MASS GLOBAL MATRIX FOR BEAM ELEMENTS

        Dim Imats, ii, Idofn As Integer
        Dim BeamMass(2) As Single 'element mass at nodes i, j
        Dim Lengt, Leng2, Delt1(NDIME), Mmat(12, 12), Trasf(12, 12), Sum, Cons1, Cons2 As
Double
        Dim Sa(12, 24), Asa(24, 24) As Double 'same matrices as for Stiffness matrix
        ReDim MassC(78)

        Imats = Mater(Ielem)
        If PROPS(Imats, 7) = 0 Then Exit Sub

        Leng2 = 0#
        For Idime = 1 To NDIME
```

```
            Delt1(Idime) = XYCOO(Idime + 3, Ielem) - XYCOO(Idime, Ielem)
            Leng2 = Leng2 + Delt1(Idime) * Delt1(Idime)
        Next Idime
        Lengt = Math.Sqrt(Leng2)

        BeamMass(1) = PROPS(Imats, 7) * PROPS(Imats, 2) * Lengt / 2
        BeamMass(2) = BeamMass(1)
        Cons1 = PROPS(Imats, 7) * PROPS(Imats, 2) * Lengt / 420
        Cons2 = PROPS(Imats, 8) / PROPS(Imats, 2) * 70

        If IfLumped Then
            Mmat(1, 1) = Cons1 * 210
            Mmat(2, 2) = Mmat(1, 1)
            Mmat(3, 3) = Mmat(1, 1)
            Mmat(7, 7) = Mmat(1, 1)
            Mmat(8, 8) = Mmat(1, 1)
            Mmat(9, 9) = Mmat(1, 1)

            'lumped matrix is already global and is stored in a vector according to Bathe
technique in STAP program (only diagonal elements)
            For Ievab = 1 To NEVAB
                MassC(Ievab) = Mmat(Ievab, Ievab)
            Next Ievab
        Else
            'consistent 3-D mass matrix

            'Mjj
            Mmat(1, 1) = Cons1 * 140
            Mmat(2, 2) = Cons1 * 156
            Mmat(2, 6) = Cons1 * 22 * Lengt
            Mmat(3, 3) = Cons1 * 156
            Mmat(3, 5) = -Cons1 * 22 * Lengt
            Mmat(4, 4) = 0 'Cons2 * 2 '0
            Mmat(5, 3) = Mmat(3, 5)
            Mmat(5, 5) = Cons1 * 4 * Leng2
            Mmat(6, 2) = Mmat(2, 6)
            Mmat(6, 6) = Mmat(5, 5)

            'Mkk
            For Idofn = 1 To NDOFN
                Mmat(Idofn + NDOFN, Idofn + NDOFN) = Mmat(Idofn, Idofn)
            Next Idofn

            Mmat(2 + NDOFN, 6 + NDOFN) = -Mmat(2, 6)
            Mmat(3 + NDOFN, 5 + NDOFN) = -Mmat(3, 5)
            Mmat(5 + NDOFN, 3 + NDOFN) = -Mmat(5, 3)
            Mmat(6 + NDOFN, 2 + NDOFN) = -Mmat(6, 2)

            'Mkj
            Mmat(1 + NDOFN, 1) = Cons1 * 70
            Mmat(2 + NDOFN, 2) = Cons1 * 54
            Mmat(2 + NDOFN, 6) = Cons1 * 13 * Lengt
            Mmat(3 + NDOFN, 3) = Cons1 * 54
            Mmat(3 + NDOFN, 5) = -Cons1 * 13 * Lengt
            Mmat(4 + NDOFN, 4) = 0 'Cons2 '0
            Mmat(5 + NDOFN, 3) = -Mmat(3 + NDOFN, 5)
            Mmat(5 + NDOFN, 5) = -Cons1 * 3 * Leng2
            Mmat(6 + NDOFN, 2) = -Mmat(2 + NDOFN, 6)
            Mmat(6 + NDOFN, 6) = -Cons1 * 3 * Leng2

            'Mjk
            For Irow = 1 + NDOFN To NDOFN + NDOFN
                For Icol = 1 To NDOFN
                    Mmat(Icol, Irow) = Mmat(Irow, Icol)
                Next Icol
            Next Irow

            'if lumped matrix is requested, don't make [T]T * [M] * [T] because global
and local matrices are the same
            Call FillTrasf(Ielem, Trasf) 'fill Trasf(12, 12) matrix with 4 Transformation
Tmat(6, 6)
```

```
            'first product: Sa(12, 12) = [T]transp * [Mmat]
            'change rows and columns of Tmat to obtain the transpose matrix
            For Ievab = 1 To NEVAB
                For Jevab = 1 To NEVAB
                    Sum = 0
                    For Kevab = 1 To NEVAB
                        Sum += Trasf(Kevab, Ievab) * Mmat(Kevab, Jevab)
                    Next Kevab
                    Sa(Ievab, Jevab) = Sum
                Next Jevab
            Next Ievab

            'second product: Asa(12, 12) = [Sa] * [T]
            For Ievab = 1 To NEVAB
                For Jevab = 1 To NEVAB
                    Sum = 0
                    For Kevab = 1 To NEVAB
                        Sum += Sa(Ievab, Kevab) * Trasf(Kevab, Jevab)
                    Next Kevab
                    Asa(Ievab, Jevab) = Sum
                Next Jevab
            Next Ievab

            'store global mass matrix in a vector according to Bathe technique in STAP
program
            ii = 0
            For Irow = 1 To 12
                For Icol = Irow To 12
                    ii += 1
                    MassC(ii) = Asa(Irow, Icol)
                Next Icol
            Next Irow
        End If

        Call ADDBAN_Mass(Ielem, BeamMass)

    End Sub
```

```
    Sub ADDBAN_Mass(Ielem As Integer, BeamMass() As Single)

        Dim N1, Idofn, Jdofn, Ievab, MAADD, Kevab, IJdif, Inde1, Inde2 As Integer

        N1 = 0

        'calculate masses on not retrained dof
        For Ievab = 1 To NEVAB
            Idofn = LMDOF(Ievab, Ielem)
            If Idofn <> 0 Then
                If Ievab <= NDOFN Then
                    TotalMass(Ievab) += BeamMass(1)
                    If Ievab = 1 Then NodalMass(Inc1(Ielem)) += BeamMass(1)
                Else
                    TotalMass(Ievab - NDOFN) += BeamMass(2)
                    If Ievab = NDOFN + 1 Then NodalMass(Inc2(Ielem)) += BeamMass(2)
                End If
            End If
        Next Ievab

        If IfLumped Then
            'lumped mass matrix
            For Ievab = 1 To NEVAB
                Idofn = LMDOF(Ievab, Ielem)
                If Idofn <> 0 Then
                    GLOBM(Idofn) += MassC(Ievab)
                End If
            Next Ievab
        Else
            'consistent mass matrix
```

```
            For Ievab = 1 To NEVAB
                Idofn = LMDOF(Ievab, Ielem)

                If Idofn > 0 Then
                    MAADD = MAXAD(Idofn)
                    Kevab = Ievab

                    For Jevab = 1 To NEVAB
                        Jdofn = LMDOF(Jevab, Ielem)
                        IJdif = Idofn - Jdofn

                        If Jdofn > 0 And IJdif >= 0 Then
                            Inde1 = MAADD + IJdif
                            Inde2 = Kevab
                            If Jevab >= Ievab Then Inde2 = Jevab + N1
                            GLOBM(Inde1) += MassC(Inde2)
                        End If
                        Kevab = Kevab + NEVAB - Jevab
                    Next Jevab
                End If
                N1 = N1 + NEVAB - Ievab
            Next Ievab
        End If

    End Sub
```

```
    Sub Write1_Eigen(File1 As StreamWriter)

        Dim Dumm$ = "", Spc$ = ""

        PrtString = Spc$ + "=====================================================" + vbCrLf
        PrtString += Spc$ + "          M D F E M  -  DYNAMIC RESPONSE MODULE       " + 
vbCrLf
        PrtString  +=  Spc$  +  "=====================================================" + 
vbCrLf

        PrtString += vbCrLf : PrtString += vbCrLf
        PrtString += Spc$ + "***  G E N E R A L   D A T A  ***" + vbCrLf
        PrtString += vbCrLf

        PrtString  +=  Spc$  +  "N.  OF  POINTS  .............................  ="  + 
String.Format("{0,5}", Npoin.ToString("####0")) + vbCrLf
        PrtString  +=  Spc$  +  "N.  OF  ELEMENTS  ...........................  ="  + 
String.Format("{0,5}", Nelem.ToString("####0")) + vbCrLf

        PrtString  +=  Spc$  +  "N.  OF  EIGENVALUES  REQUESTED  ..............  ="  + 
String.Format("{0,5}", NROOT.ToString("####0")) + vbCrLf
        Dumm$ = "Consistent"
        If IfLumped Then Dumm$ = "Lumped"
        PrtString += Spc$ + "TYPE OF MASS MATRICES .................... :     " + Dumm$ + 
vbCrLf
        PrtString  +=  Spc$  +  "N.  OF  SUBSPACE  ITERATIONS  ................  ="  + 
String.Format("{0,5}", NITEM.ToString("####0")) + vbCrLf
        PrtString  +=  Spc$  +  "TOLERANCE  FOR  EIGENVALUES  CONVERGENCE  CHECK  ="  + 
String.Format("{0,11}", RTOL.ToString("0.0000E+00")) + vbCrLf
        Dumm$ = "No"
        If IFSS = 1 Then Dumm$ = "Yes"
        PrtString  +=  Spc$  +  "STURM  SEQUENCE  CHECK  .....................  :  "  + 
String.Format("{0,4}", Dumm$) + vbCrLf
        Dumm$ = "No"
        If IFPR = 1 Then Dumm$ = "Yes"
        PrtString  +=  Spc$  +  "PRINT  INTERMEDIATE  ITERATION  RESULTS  ......  :  "  + 
String.Format("{0,4}", Dumm$) + vbCrLf
        PrtString += vbCrLf
        PrtString += Spc$ + "for the static analysis refer to the xxxx.dat file" + vbCrLf
        PrtString += vbCrLf
        If Not IfLumped Then
            PrtString += Spc$ + "IMPORTANT REMARK!" + vbCrLf
```

```
            PrtString += Spc$ + "If consistent mass matrix is requested (see above), torsional modes are neglected." + vbCrLf
            PrtString += Spc$ + "To modify this assumption, some changes in Sub BeamMass are needed." + vbCrLf
        End If

        PrtString += vbCrLf
        File1.WriteLine(PrtString)

    End Sub
```

```
    Sub Write2_Eigen(File1 As StreamWriter)

        Dim Spc$ = ""

        PrtString = Spc$ + "* * *  M A T E R I A L   P R O P E R T I E S  * * *" + vbCrLf
        PrtString += vbCrLf

        PrtString += Spc$ + "  Set    mass density" + vbCrLf

        ' *** LOOP OVER MATERIAL SETS

        For Isets = 1 To Nmats
            PrtString += Spc$ + String.Format("{0,5}", Isets.ToString("####0")) + "   "

            PrtString += String.Format("{0,12}", PROPS(Isets, 7).ToString("0.0000E+00")) + vbCrLf
        Next Isets
        File1.WriteLine(PrtString)

    End Sub
```

```
    Sub AddLoadMass(Icase As Integer)
        Dim Ldofn, Lkglo As Long
        Dim p1, p2, Mass, Mass_i, Mass_j As Double
        Dim Delt1(3), Leng2, Lengt As Double

        If Gaccel = 0 Then Exit Sub

        'add point loads mass: PointLoad array contains loads referred to the global system
        For Ipoin = 1 To Npoin
            If Ntype = 2 Or Ntype = 3 Then
                Mass = Math.Abs(PointLoad(Icase, Ipoin, 1) / Gaccel) 'Fx
                Mass += Math.Abs(PointLoad(Icase, Ipoin, 2) / Gaccel) 'Fy
                Mass += Math.Abs(PointLoad(Icase, Ipoin, 3) / Gaccel) 'Fz

                If Mass > 0 Then
                    NodalMass(Ipoin) += Mass
                    For Idofn = 1 To NDOFN
                        Ldofn = IDDOF(Idofn, Ipoin)
                        If Ldofn <> 0 Then
                            TotalMass(Idofn) += Mass 'nodal masses can act in x, y, z directions and are effective masses for all dof
                        End If
                    Next Idofn

                    If IfLumped Then
                        Lkglo = IDDOF(1, Ipoin)
                        GLOBM(Lkglo) += Mass
                        Lkglo = IDDOF(2, Ipoin)
                        GLOBM(Lkglo) += Mass
                        Lkglo = IDDOF(3, Ipoin)
                        GLOBM(Lkglo) += Mass
                    Else
                        Lkglo = MAXAD(IDDOF(1, Ipoin))
                        GLOBM(Lkglo) += Mass
```

```
                        Lkglo = MAXAD(IDDOF(2, Ipoin))
                        GLOBM(Lkglo) += Mass
                        Lkglo = MAXAD(IDDOF(3, Ipoin))
                        GLOBM(Lkglo) += Mass
                    End If
                End If
            End If
        Next Ipoin

        'add linear loads mass: Eload array contains loads already tranformed from the
local system to the global system
        For Ielem = 1 To Nelem
            Leng2 = 0
            For Idime = 1 To NDIME
                Delt1(Idime) = XYCOO(Idime + 3, Ielem) - XYCOO(Idime, Ielem)
                Leng2 += Delt1(Idime) * Delt1(Idime)
            Next Idime
            Lengt = Math.Sqrt(Leng2)

            If Ntype = 2 Or Ntype = 3 Then
                p1 = (Dload(Icase, Ielem, 1) + Dload(Icase, Ielem, 2) + Dload(Icase,
Ielem, 3)) / Gaccel 'qx-i + qy-i + qz-i
                p2 = (Dload(Icase, Ielem, 4) + Dload(Icase, Ielem, 5) + Dload(Icase,
Ielem, 6)) / Gaccel 'qx-j + qy-j + qz-j

                Mass_i = Math.Abs(Lengt * (p1 / 2 + (p2 - p1) / 6))

                If Mass_i > 0 Then
                    NodalMass(Inc1(Ielem)) += Mass_i

                    For Idofn = 1 To NDOFN
                        Ldofn = IDDOF(Idofn, Inc1(Ielem))
                        If Ldofn <> 0 Then
                            TotalMass(Idofn) += Mass_i 'linear loads masses can act in x,
y, z directions and are effective masses for all dof
                        End If
                    Next Idofn

                    If IfLumped Then
                        Lkglo = IDDOF(1, Inc1(Ielem))
                        GLOBM(Lkglo) += Mass_i
                        Lkglo = IDDOF(2, Inc1(Ielem))
                        GLOBM(Lkglo) += Mass_i
                        Lkglo = IDDOF(3, Inc1(Ielem))
                        GLOBM(Lkglo) += Mass_i
                    Else
                        Lkglo = MAXAD(IDDOF(1, Inc1(Ielem)))
                        GLOBM(Lkglo) += Mass_i
                        Lkglo = MAXAD(IDDOF(2, Inc1(Ielem)))
                        GLOBM(Lkglo) += Mass_i
                        Lkglo = MAXAD(IDDOF(3, Inc1(Ielem)))
                        GLOBM(Lkglo) += Mass_i
                    End If
                End If

                Mass_j = Math.Abs(Lengt * (p1 / 2 + (p2 - p1) / 3))

                If Mass_j > 0 Then
                    NodalMass(Inc2(Ielem)) += Mass_j

                    For Idofn = 1 To NDOFN
                        Ldofn = IDDOF(Idofn, Inc2(Ielem))
                        If Ldofn <> 0 Then
                            TotalMass(Idofn) += Mass_j 'linear loads masses can act in x,
y, z directions and are effective masses for all dof
                        End If
                    Next Idofn

                    If IfLumped Then
                        Lkglo = IDDOF(1, Inc2(Ielem))
                        GLOBM(Lkglo) += Mass_j
```

```
                        Lkglo = IDDOF(2, Inc2(Ielem))
                        GLOBM(Lkglo) += Mass_j
                        Lkglo = IDDOF(3, Inc2(Ielem))
                        GLOBM(Lkglo) += Mass_j
                    Else
                        Lkglo = MAXAD(IDDOF(1, Inc2(Ielem)))
                        GLOBM(Lkglo) += Mass_j
                        Lkglo = MAXAD(IDDOF(2, Inc2(Ielem)))
                        GLOBM(Lkglo) += Mass_j
                        Lkglo = MAXAD(IDDOF(3, Inc2(Ielem)))
                        GLOBM(Lkglo) += Mass_j
                    End If
                End If
            End If
        Next Ielem

    End Sub
```

The subroutine Sspace is represented because there are some differences from the one listed in Appendix A, where the Response Spectrum Analysis is not performed.

```
    Sub Sspace(File1 As StreamWriter)
        '--------------------------------------------------------------------------------
--------------
        ' Program to solve for the smallest eigenvalues and corresponding eigenvectors
        ' in the generalized eigenproblem using the subspace iteration method
        '
        ' written by K. J. Bathe: "Finite Element Procedures in Engineering Analysis"
        '                         Prentice-Hall, 1982
        '
        ' revised and translated in vb.net by Paolo Varagnolo, 2020
        '
        ' Input variables
        '--------------------------------------------------------------------------------
--------------
        ' GLOBK(NKGLO)     = stiffness matrix in compacted form
        '                    (global scope variable, already assembled)
        ' GLOBM(NMGLO)     = mass matrix in compacted form
        '                    (global scope variable, already assembled)
        ' MAXAD(NDOFT + 1) = vector containing the addresses of diagonal elements of
GLOBK()
        '                    (global scope variable, already assembled)
        ' R(NDOFT, NC)     = eigenvectors on solution exit
        ' EIGV(NC)         = eigenvalues on solution exit
        ' TT(NDOFT)        = working vector
        ' W(NDOFT)         = working vector
        ' AR(NNC)          = working vector storing projection of GLOBK
        ' BR(NNC)          = working vector storing projection of GLOBM
        ' VEC(NC, NC)      = working matrix
        ' D(NC)            = working vector
        ' RTOLV(NC)        = working vector
        ' BUP(NC)          = working vector
        ' BLO(NC)          = working vector
        ' BUPC(NC)         = working vector
        '
        ' NKGLO            = number of elements below skyline of GLOBK()
        '                    (global scope variable, already assigned)
        ' NMGLO            = number of elements below skyline of GLOBM()
        '                    (global scope variable, already assigned)
        ' NDOFT          = total number of degrees of freedom = order of GLOBK(), GLOBM()
        '                    (global scope variable, already assigned)
        ' NC              = number of iteration vectors used, usually set to MIN(2*NROOT,
NROOT + 8),
        '                   cannot be larger then the number of mass degrees of freedom
        ' NNC              = NC * (NC + 1) / 2 dimension of storage vectors AR, BR
        ' NROOT            = number of required eigenvalues and eigenvectors
        '                    (global scope variable, already assigned)
        ' RTOL             = convergence tolerance on eigenvalues (1E-6 or smaller)
        '                    (global scope variable, already assigned)
```

```
        ' NITEM            = maximum number of subspace iterations permitted (usually set
to 16)
        '                    the parameters NC and/or NITEM must be increased
        '                    if a Then solutions has Not converged
        '                    (global scope variable, already assigned)
        ' NSMAX            = maximum number of sweeps in Jacobi iteration
        ' IFSS              = flag for Sturm sequence check:  = 0 --> no check; = 1 -->
check
        '                    (global scope variable, already assigned)
        ' IFPR              = flag for intermediate printing: = 0 --> no check; = 1 -->
check
        '                    (global scope variable, already assigned)
        ' Dstif            = scratch streamwriter to store stiffness matrix
        ' File1            = streamwriter for output file
        '--------------------------------------------------------------------------------
--------------
        ' Output variables
        '--------------------------------------------------------------------------------
--------------
        ' EIGV(NROOT)      = eigenvalues
        ' R(NDOFT, NROOT)  = eigenvectors
        '--------------------------------------------------------------------------------
--------------
        Dim NC, MassDOF, NNC, ij, Iconv, NSCH, NSMAX, N1, NC1, ND, ISH, Nite, Nei, Icoun,
Idofn As Integer
        Dim itemp As Integer
        Dim i%, j%, l%, ii%
        Dim TOLJ, RT, MaxTol, ART, BRT, Dummy, Vnorm, Wnorm, Shift As Double
        Dim Text As String
        Dim ConvReached, Swapped As Boolean

        MassDOF = CalculateMassDOF()
        NC = Math.Min(2 * NROOT, NROOT + 8)
        NC = Math.Min(NC, MassDOF)
        NNC = NC * (NC + 1) / 2

        Dim R(NDOFT, NC), TT(NDOFT), W(NDOFT), EIGV(NC), D(NC), VEC(NC, NC), AR(NNC),
BR(NNC) As Double
        Dim RTOLV(NC), BUP(NC), BLO(NC), BUPC(NC) As Double

        TOLJ = 0.000000000001 'TOLERANCE FOR jACOBI ITERATION

        If NROOT > NDOFT Then
            Text = "The number of requested eigenvalues is greater then " + vbCrLf
            Text += "the number of degrees of freedom in the model." + vbCrLf
            Text += "Only " + NDOFT.ToString + " eigenvalues will be serched." + vbCrLf
            MsgBox(Text, vbExclamation, "Warning")
            File1.WriteLine(Text)
            NROOT = NDOFT
        End If

        If NROOT > MassDOF Then
            Text = "The number of requested eigenvalues is greater then " + vbCrLf
            Text += "the number of mass degrees of freedom." + vbCrLf
            Text += "Only " + MassDOF.ToString + " eigenvalues will be serched." + vbCrLf
            MsgBox(Text, vbExclamation, "Warning")
            File1.WriteLine(Text)
            NROOT = MassDOF
        End If

        If IFPR <> 0 Then
            Text = vbCrLf
            Text += "global stiffness matrix in compacted form " + vbCrLf
            For i% = 1 To NKGLO
                Text += String.Format("{0,15}", GLOBK(i%).ToString("0.00000000E+00")) +
vbCrLf
            Next i%
            File1.WriteLine(Text)

            Text = vbCrLf
            Text += "global mass matrix in compacted form " + vbCrLf
```

```
            For i% = 1 To NMGLO
                Text += String.Format("{0,15}", GLOBM(i%).ToString("0.00000000E+00")) +
vbCrLf
            Next i%
            File1.WriteLine(Text)
        End If

        'Initialization
        Iconv = 0
        NSCH = 0
        NSMAX = 12
        N1 = NC + 1
        NC1 = NC - 1

        Dim Dstif As System.IO.StreamWriter = Nothing
        Dim FileWork0 As String = myPath + "WORK0" 'open a file where will be saved the
stiffness global matrix
        Dstif = My.Computer.FileSystem.OpenTextFileWriter(FileWork0, True)

        'write global stiffness matrix
        For i% = 1 To NKGLO
            Dstif.WriteLine(GLOBK(i%))
        Next i%
        Dstif.Close()

        ReDim D(NC), R(NDOFT, NC) 'this set the arrays to zero

        'establish starting iteration vectors
        ND = Int(NDOFT / NC)
        If NMGLO <= NDOFT Then
            j% = 0
            For i% = 1 To NDOFT
                ii% = MAXAD(i%)
                R(i%, 1) = GLOBM(i%)
                If GLOBM(i%) > 0 Then j% += 1
                W(i%) = GLOBM(i%) / GLOBK(ii%)
            Next i%
            If NC > j% Then
                Text = "The number of iteration vectors must not exceed the number of mass
degrees of freedom." + vbCrLf
                MsgBox(Text, vbExclamation, "Warning")
                File1.WriteLine(Text)
                End
            End If
        Else
            For i% = 1 To NDOFT
                ii% = MAXAD(i%)
                R(i%, 1) = GLOBM(ii%)
                W(i%) = GLOBM(ii) / GLOBK(ii)
            Next i%
        End If

        l% = NDOFT - ND
        For j% = 2 To NC
            RT = 0
            For i% = 1 To l%
                If W(i%) >= RT Then
                    RT = W(i%)
                    ij = i%
                End If
            Next i%
            For i% = l% To NDOFT
                If W(i%) > RT Then
                    RT = W(i%)
                    ij = i%
                End If
            Next i%
            TT(j%) = ij
            W(ij) = 0.0
            l% -= ND
            R(ij, j%) = 1.0
```

```
        Next j%

        PrtString = "Degrees of freedom excited by unit starting iteration vectors" + 
vbCrLf
        Call PrintArray_1_10_Int(PrtString, TT, 2, NC)
        File1.WriteLine(PrtString)

        'factorize matrix GLOBK() into (L)*(D)*(L(T))
        ISH = 0
        Call Decomp(ISH, File1)
        If Errore Then
            File1.Close() : Exit Sub
        End If

        'start of iteration loop
        Nite = 0
        ConvReached = False
        Do While Iconv = 0 'it is an infinite loop, Iconv remains = 0. The exit from the 
loop occurs when ConvReached becomes true
            Nite += 1
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Iteration number: " + String.Format("{0,4}", 
Nite.ToString("###0")) + vbCrLf
                File1.WriteLine(Text)
            End If

            'calculate the projection of GLOBK and GLOBM
            ij = 0
            For j% = 1 To NC
                For k% = 1 To NDOFT
                    TT(k%) = R(k%, j%)
                Next k%
                Call REDBAK(TT, File1)
                For i% = j% To NC
                    ART = 0
                    For k% = 1 To NDOFT
                        ART += R(k%, i%) * TT(k%)
                    Next k%
                    ij += 1
                    AR(ij) = ART
                Next i%
                For k% = 1 To NDOFT
                    R(k%, j%) = TT(k%)
                Next k%
            Next j%

            If IFPR <> 0 Then
                PrtString = vbCrLf
                PrtString += "array TT() after REDBAK" + vbCrLf
                Call PrintArray_1_10_Real(PrtString, TT, 1, NDOFT)
                File1.WriteLine(PrtString)
            End If

            ij = 0
            For j% = 1 To NC
                Call MULT(TT, GLOBM, R, j%, NMGLO)
                For i% = j% To NC
                    BRT = 0
                    For k% = 1 To NDOFT
                        BRT += R(k%, i%) * TT(k%)
                    Next k%
                    ij += 1
                    BR(ij) = BRT
                Next i%
                If Not ConvReached Then
                    For k% = 1 To NDOFT
                        R(k%, j%) = TT(k%)
                    Next k%
                End If
            Next j%
```

```
            'solve for eigensystem of subspace operators
            If IFPR <> 0 Then
                Call PrintProjections(AR, BR, NC, File1)
            End If

            Call Jacobi(AR, BR, VEC, EIGV, W, NC, NNC, TOLJ, NSMAX, IFPR, File1)
            If Errore Then
                FileClose() : Exit Sub
            End If

            If IFPR <> 0 Then
                Text = "AR and BR after Jacobi diagonalization"
                File1.WriteLine(Text)
                Call PrintProjections(AR, BR, NC, File1)
            End If

            'arrange eigenvalues in ascending order
            Swapped = True
            Do Until Swapped = False
                Swapped = False
                ii = 1
                For i% = 1 To NC1
                    itemp = ii + N1 - i%
                    If EIGV(i% + 1) < EIGV(i%) Then
                        Swapped = True
                        Dummy = EIGV(i% + 1)
                        EIGV(i% + 1) = EIGV(i%)
                        EIGV(i%) = Dummy
                        Dummy = BR(itemp)
                        BR(itemp) = BR(ii)
                        BR(ii) = Dummy
                        For k% = 1 To NC
                            Dummy = VEC(k%, i% + 1)
                            VEC(k%, i% + 1) = VEC(k%, i%)
                            VEC(k%, i%) = Dummy
                        Next k%
                    End If
                    ii = itemp
                Next i%
            Loop

            If IFPR <> 0 Then
                Text = "Eigenvalues of AR - Lambda * BR" + vbCrLf
                Call PrintArray_1_10_Real(Text, EIGV, 1, NC)
                File1.WriteLine(Text)
            End If

            'calculate GLOBM times approximate or final eigenvectors
            For i% = 1 To NDOFT
                For j% = 1 To NC
                    TT(j%) = R(i%, j%)
                Next j%
                For k% = 1 To NC
                    RT = 0
                    For l% = 1 To NC
                        RT += TT(l%) * VEC(l%, k%)
                    Next l%
                    R(i%, k%) = RT
                Next k%
            Next i%

            '------------------------------------------------------------------------------
-------------
            'this is the real exit from the iteration loop
            If ConvReached Then Exit Do
            '------------------------------------------------------------------------------
-------------

            'check for convergence of eigenvalues
            For i% = 1 To NC
```

```
                RTOLV(i%) = Math.Abs(EIGV(i%) - D(i%)) / EIGV(i%)
            Next i%
            If IFPR <> 0 Then
                Text = vbCrLf
                Text += "Relative tolerance reached on eigenvalues." + vbCrLf
                Call PrintArray_1_10_Real(Text, RTOLV, 1, NC)
                File1.WriteLine(Text)
            End If

            MaxTol = -9999
            For i% = 1 To NROOT
                If MaxTol < RTOLV(i%) Then MaxTol = RTOLV(i%)
            Next i%
            If MaxTol < RTOL Then
                'convergence reached
                Text = vbCrLf
                Text += "Convergence reached for tolerance = " + String.Format("{0,12}",
RTOL.ToString("0.00000E+00")) '+ vbCrLf
                File1.WriteLine(Text)
                ConvReached = True 'Iconv = 1
            Else
                If Nite >= NITEM Then
                    'convergence not reached
                    Text = vbCrLf
                    Text += "No convergence in maximum number of iteratioons permitted."
+ vbCrLf
                    Text += "Current iteration values will be accepted." + vbCrLf
                    Text += "The Sturm sequence check is not performed." + vbCrLf
                    File1.WriteLine(Text)
                    ConvReached = True 'Iconv = 2
                    IFSS = 0
                Else
                    For i% = 1 To NC
                        D(i%) = EIGV(i%)
                    Next i%
                End If
            End If
        Loop 'end of iteration loop

        Text = vbCrLf
        Text += "The calculated eigenvalues are:" + vbCrLf
        Call PrintArray_1_10_Real(Text, EIGV, 1, NROOT)
        File1.WriteLine(Text)

        Text = ""
        Text += "The calculated eigenvectors are:" + vbCrLf
        For Iroot = 1 To NROOT
            Dim Dumm(NDOFT) As Double
            For Idofn = 1 To NDOFT
                Dumm(Idofn) = R(Idofn, Iroot)
            Next Idofn
            Call PrintArray_1_10_Real(Text, Dumm, 1, NDOFT)
        Next Iroot
        File1.WriteLine(Text)

        'calculate and print error norms

        'read global stiffness matrix
        Using sr As StreamReader = File.OpenText(FileWork0) 'stiffness global matrix
            For i% = 1 To NKGLO
                GLOBK(i%) = sr.ReadLine
            Next i%
        End Using

        For l% = 1 To NROOT
            RT = EIGV(l%)
            Call MULT(TT, GLOBK, R, l%, NKGLO)
            Vnorm = 0
            For i% = 1 To NDOFT
                Vnorm += TT(i%) * TT(i%)
            Next i%
```

```
            Call MULT(W, GLOBM, R, l%, NMGLO)
            Wnorm = 0
            For i% = 1 To NDOFT
                TT(i%) -= RT * W(i%)
                Wnorm += TT(i%) * TT(i%)
            Next i%
            Vnorm = Math.Sqrt(Vnorm)
            Wnorm = Math.Sqrt(Wnorm)
            D(l%) = Wnorm / Vnorm
        Next l%

        If IFPR > 0 Then
            Text = vbCrLf
            Text += "Print error norms on the eigenvalues" + vbCrLf
            Call PrintArray_1_10_Real(Text, D, 1, NROOT)
            File1.WriteLine(Text)
        End If

        'apply Sturm sequence check
        If IFSS <> 0 Then 'IFSS is the flag for Sturm sequence check
            Call SturmCheck(EIGV, RTOLV, BUP, BLO, BUPC, D, NC, Nei, RTOL, Shift, File1)
            If Errore Then
                File1.Close() : Exit Sub
            End If
            If IFPR > 0 Then
                Text = vbCrLf
                Text   +=   "Check   applied   at   shift:   "   +   String.Format("{0,12}",
Shift.ToString("0.00000E+00")) '+ vbCrLf
                File1.WriteLine(Text)
            End If

            'shift matrix GLOBK
            'read global stiffness matrix
            Using sr As StreamReader = File.OpenText(FileWork0) 'stiffness global matrix
file
                For i% = 1 To NKGLO
                    GLOBK(i%) = sr.ReadLine
                Next i%
            End Using

            If NMGLO <= NDOFT Then
                For i% = 1 To NDOFT
                    ii = MAXAD(i%)
                    GLOBK(ii) -= GLOBM(i%) * Shift
                Next i%
            Else
                For i% = 1 To NKGLO
                    GLOBK(i%) -= GLOBM(i%) * Shift
                Next i%
            End If

            'factorize shifted matrix
            ISH = 1
            Call Decomp(ISH, File1)
            If Errore Then
                File1.Close() : Exit Sub
            End If

            'count number of negative diagonal elements
            NSCH = 0
            For i% = 1 To NDOFT
                ii = MAXAD(i%)
                If GLOBK(ii) < 0 Then NSCH += 1
            Next i%

            If NSCH = Nei Then
                Text = ""
                Text   +=   "We   found   the   lowest:   "   +   String.Format("{0,4}",
NSCH.ToString("###0")) + " eigenvalues "
                Text += "(" + NROOT.ToString + " had to be found)" + vbCrLf
                File1.WriteLine(Text)
```

```
            Else
                Text = ""
                Text += "There are: " + String.Format("{0,4}", (NSCH -
Nei).ToString("###0")) + " eigenvalues missing" + vbCrLf
                File1.WriteLine(Text)
            End If
        End If 'Sturm sequence check

        'Write FREQUENCIES (ADDED BY PV)
        If IFSS = 0 Then
            Text = vbCrLf + " PRINT OF FREQUENCIES" + vbCrLf
        Else
            Text = " PRINT OF FREQUENCIES" + vbCrLf
        End If
        Text += vbCrLf
        Text += " MODE          CIRCULAR " + vbCrLf
        Text += " NUMBER        FREQUENCY        FREQUENCY        PERIOD" + vbCrLf
        Text += "              (RAD/SEC)       (CYCLES/SEC)       (SEC)" + vbCrLf
        Text += "-------------------------------------------------------------"
        File1.WriteLine(Text)

        ReDim CircFreq(NROOT), Frequency(NROOT), Period(NROOT)
        Dim TPI As Double = 8 * Math.Atan(1)
        For i% = 1 To NROOT
            CircFreq(i%) = Math.Sqrt(EIGV(i%)) 'circular frequency
            Frequency(i%) = CircFreq(i%) / TPI 'frequency
            Period(i%) = TPI / CircFreq(i%) 'period
            Text = String.Format("{0,5}", i%.ToString("####0")) + "    "
            Text += String.Format("{0,17}", CircFreq(i%).ToString("E8"))
            Text += String.Format("{0,17}", Frequency(i%).ToString("E8"))
            Text += String.Format("{0,17}", Period(i%).ToString("E8"))
            File1.WriteLine(Text)
        Next i%

        CurCase = Ncase + NCOMB
        Text = vbCrLf
        Text += "Nodes displacements / rotations" + vbCrLf
        Text += "-------------------------------" + vbCrLf
        Text += "  Node    eigen" + vbCrLf
        Text += " number   vector   X-displac.   Y-displac.   Z-displac.  XX-rotation  YY-
rotation  ZZ-rotation" '+ vbCrLf
        File1.WriteLine(Text)

        For Iroot = 1 To NROOT
            CurCase += 1
            Icoun = 0
            For Ipoin = 1 To Npoin
                For Idofn = 1 To NDOFN
                    If IDDOF(Idofn, Ipoin) > 0 Then
                        Icoun += 1
                        Displ(CurCase, Ipoin, Idofn) = R(Icoun, Iroot)
                    End If
                Next Idofn
            Next Ipoin
        Next Iroot

        'print of eigenvalues
        For Ipoin = 1 To Npoin
            CurCase = Ncase + NCOMB
            Text = String.Format("{0,6}", Ipoin.ToString("#####0"))
            For Iroot = 1 To NROOT
                CurCase += 1
                If Iroot > 1 Then Text = "      "
                Text += String.Format("{0,8}", Iroot.ToString("#####0")) + "  "
                For Idofn = 1 To NDOFN
                    Text += String.Format("{0,13}", Displ(CurCase, Ipoin,
Idofn).ToString("0.00000E+00"))
                Next Idofn
                File1.WriteLine(Text)
            Next Iroot
        Next Ipoin
```

```
        Call ParticipantMassesCalculation(R) 'calculate participant masses for seismic
analysis
        Call WriteParticipantMasses(File1)

        If Ifspe > 0 Then
            Call SpectralResponse(R) 'calculate Modal Response Spectrum analysis
            Call WriteSpectrumDisp(File1)
        End If

        If Dir(myPath + "WORK0") <> "" Then Kill(myPath & "WORK0")

    End Sub
```

```
    Sub WriteParticipantMasses(File1 As StreamWriter)

        Dim Text As String
        Dim ParticTotMass(NDOFN) As Single

        'PartFact(Iroot, Icoun)

        Text = vbCrLf
        Text += "Modal participation factors" + vbCrLf
        Text += "---------------------------------------------" + vbCrLf
        Text += "  Mode   X direction Y direction Z direction" + vbCrLf
        Text += "---------------------------------------------"
        File1.WriteLine(Text)

        For Imode = 1 To NROOT
            Text = String.Format("{0,5}", Imode.ToString("####0"))
            For Idime = 1 To NDIME
                Text         +=         String.Format("{0,12}",          PartFact(Imode,
Idime).ToString("0.000000"))
            Next Idime
            File1.WriteLine(Text)
        Next Imode
        File1.WriteLine()

        Text = ""
        Text += "Modal participating mass ratio (p.m.r.)" + vbCrLf
        Text += "-------------------------------------------" + vbCrLf
        Text += "  Mode    p.m.r. X    p.m.r. Y    p.m.r. Z" + vbCrLf
        Text += "-------------------------------------------"
        File1.WriteLine(Text)

        For Imode = 1 To NROOT
            Text = String.Format("{0,5}", Imode.ToString("####0"))
            For Idofn = 1 To NDIME
                ParticTotMass(Idofn) += PartMas(Imode, Idofn)
                Text         +=         String.Format("{0,12}",          PartMas(Imode,
Idofn).ToString("0.000000"))
            Next Idofn
            File1.WriteLine(Text)
        Next Imode

        Text = "-------------------------------------------" + vbCrLf
        Text += "  Sum"
        For Idofn = 1 To NDIME
            Text += String.Format("{0,12}", ParticTotMass(Idofn).ToString("0.000000"))
        Next Idofn
        File1.WriteLine(Text)
        File1.WriteLine()

        Text = ""
        If ParticTotMass(1) < 0.85 Then
            Text  +=  "Warning!  participating  masses  in  X  direction  <  85%.  §7.3.3.1
prescriptions not observed." + vbCrLf
        Else
```

```
            Text += "Participating masses in X direction >= 85%, as prescribed in NTC 2018 §7.3.3.1." + vbCrLf
        End If
        File1.WriteLine(Text)

        Text = ""
        If ParticTotMass(2) < 0.85 Then
            Text += "Warning! participating masses in Y direction < 85%. §7.3.3.1 prescriptions not observed." + vbCrLf
        Else
            Text += "Participating masses in Y direction >= 85%, as prescribed in NTC 2018 §7.3.3.1." + vbCrLf
        End If
        File1.WriteLine(Text)

    End Sub
```

```
    Sub WriteParticipantMasses(File1 As StreamWriter)

        Dim Text As String
        Dim ParticTotMass(NDOFN) As Single

        'PartFact(Iroot, Icoun)

        Text = vbCrLf
        Text += "Modal participation factors" + vbCrLf
        Text += "---------------------------------------------" + vbCrLf
        Text += "  Mode   X direction Y direction Z direction" + vbCrLf
        Text += "---------------------------------------------"
        File1.WriteLine(Text)

        For Imode = 1 To NROOT
            Text = String.Format("{0,5}", Imode.ToString("####0"))
            For Idime = 1 To NDIME
                Text += String.Format("{0,12}", PartFact(Imode, Idime).ToString("0.000000"))
            Next Idime
            File1.WriteLine(Text)
        Next Imode
        File1.WriteLine()

        Text = ""
        Text += "Modal participating mass ratio (p.m.r.)" + vbCrLf
        Text += "-------------------------------------------" + vbCrLf
        Text += "  Mode    p.m.r. X    p.m.r. Y    p.m.r. Z" + vbCrLf
        Text += "-------------------------------------------"
        File1.WriteLine(Text)

        For Imode = 1 To NROOT
            Text = String.Format("{0,5}", Imode.ToString("####0"))
            For Idofn = 1 To NDIME
                ParticTotMass(Idofn) += PartMas(Imode, Idofn)
                Text += String.Format("{0,12}", PartMas(Imode, Idofn).ToString("0.000000"))
            Next Idofn
            File1.WriteLine(Text)
        Next Imode

        Text = "-------------------------------------------" + vbCrLf
        Text += "  Sum"
        For Idofn = 1 To NDIME
            Text += String.Format("{0,12}", ParticTotMass(Idofn).ToString("0.000000"))
        Next Idofn
        File1.WriteLine(Text)
        File1.WriteLine()

        Text = ""
        If ParticTotMass(1) < 0.85 Then
```

```
            Text  +=  "Warning!  participating  masses  in  X  direction  <  85%.  §7.3.3.1
prescriptions not observed." + vbCrLf
        Else
            Text += "Participating masses in X direction >= 85%, as prescribed in NTC 2018
§7.3.3.1." + vbCrLf
        End If
        File1.WriteLine(Text)

        Text = ""
        If ParticTotMass(2) < 0.85 Then
            Text  +=  "Warning!  participating  masses  in  Y  direction  <  85%.  §7.3.3.1
prescriptions not observed." + vbCrLf
        Else
            Text += "Participating masses in Y direction >= 85%, as prescribed in NTC 2018
§7.3.3.1." + vbCrLf
        End If
        File1.WriteLine(Text)

    End Sub

    Sub WriteSeismicForces(File1 As StreamWriter)

        Dim Text As String

        Text = vbCrLf
        Text += "======================================================" + vbCrLf
        Text += "              RESPONSE SPECTRUM ANALYSIS" + vbCrLf
        Text += "                  (Forces Method)" + vbCrLf
        Text += "======================================================" + vbCrLf
        File1.WriteLine(Text)

        Text = "    Mode     Period     Spectral acceleration" + vbCrLf
        Text += "     no.       (s)              m/s2" + vbCrLf
        Text += "----------------------------------------" + vbCrLf
        For Imode = 1 To NROOT
            Text += String.Format("{0,7}", Imode.ToString("####0")) + "   "
            Text  +=  String.Format("{0,11}",  Period(Imode).ToString("0.0000E+00"))  +  "
"
            Text  +=  String.Format("{0,11}",  ModalAcc(Imode).ToString("0.0000E+00"))  +
vbCrLf
        Next Imode
        File1.WriteLine(Text)

        'write C.Q.C. and S.R.S.S. forces
        Text = "combinations of seismic forces" + vbCrLf
        'Text += "---------------------------------------------------------------------" +
vbCrLf
        'Text += "    Node |       C.Q.C. Force         |        S.R.S.S. Force      |" +
vbCrLf
        'Text += "         |     Fx             Fy      |      Fx             Fy     |" +
vbCrLf
        'Text += "---------------------------------------------------------------------" +
vbCrLf
        Text += "----------------------------------------" + vbCrLf
        Text += "    Node |       C.Q.C. Force         |" + vbCrLf
        Text += "         |     Fx             Fy      |" + vbCrLf
        Text += "----------------------------------------" + vbCrLf

        For Ipoin = 1 To Npoin
            Text += String.Format("{0,8}", Ipoin.ToString("####0")) + "   "
            Text        +=        String.Format("{0,11}",        SeismicForceCQC(Ipoin,
1).ToString("0.0000E+00")) + "    "
            Text        +=        String.Format("{0,11}",        SeismicForceCQC(Ipoin,
2).ToString("0.0000E+00")) + "    " + vbCrLf
            'Text       +=       String.Format("{0,11}",       SeismicForceSRSS(Ipoin,
1).ToString("0.0000E+00")) + "    "
            'Text       +=       String.Format("{0,11}",       SeismicForceSRSS(Ipoin,
2).ToString("0.0000E+00")) + vbCrLf
        Next Ipoin
        'Text += "---------------------------------------------------------------------" +
vbCrLf
```

```
        Text += "----------------------------------------" + vbCrLf
        Text += "base shear "

        Text += String.Format("{0,11}", ShearForcesCQC(1, 1).ToString("0.0000E+00")) + "
" 'ShearForcesCQC(Ipoin, Idime)
        Text += String.Format("{0,11}", ShearForcesCQC(1, 2).ToString("0.0000E+00")) + "
" + vbCrLf
        'Text += String.Format("{0,11}", ShearForcesSRSS(1, 1).ToString("0.0000E+00")) +
"    " 'ShearForcesSRSS(Ipoin, Idime)
        'Text += String.Format("{0,11}", ShearForcesSRSS(1, 2).ToString("0.0000E+00")) +
vbCrLf
        File1.WriteLine(Text)

    End Sub
```

```
    Function ElemLen(Inod1 As Integer, Inod2 As Integer) As Single

        Dim dx, dy, dz, XYlen As Single

        ElemLen = 0

        dx = Xcoor(Inod2) - Xcoor(Inod1)
        dy = Ycoor(Inod2) - Ycoor(Inod1)
        dz = Zcoor(Inod2) - Zcoor(Inod1)

        XYlen = Math.Sqrt(dx ^ 2 + dy ^ 2)
        ElemLen = Math.Sqrt(XYlen ^ 2 + dz ^ 2)

    End Function
```

```
    Sub WriteSpectrumDisp(File1 As StreamWriter)

        Dim Text, Text1, Text2 As String

        Call WriteSpectrum(File1)

        Text = vbCrLf : Text += vbCrLf
        Text += "======================================================" + vbCrLf
        Text += "             RESPONSE SPECTRUM ANALYSIS" + vbCrLf
        Text += "               (Displacements Method)" + vbCrLf
        Text += "======================================================" '+ vbCrLf
        File1.WriteLine(Text)

        'print displacements of the simple oscillator, in spectrum response
        Text = vbCrLf
        Text += "Nodes displacements / rotations" + vbCrLf
        Text += "-------------------------------" + vbCrLf
        Text += "  Node    eigen" + vbCrLf
        Text += " number   vector   X-displac.   Y-displac.   Z-displac.  XX-rotation  YY-
rotation  ZZ-rotation" '+ vbCrLf
        File1.WriteLine(Text)

        For Ipoin = 1 To Npoin
            Text = String.Format("{0,6}", Ipoin.ToString("#####0"))
            For Iroot = 1 To NROOT
                If Iroot > 1 Then Text = "      "
                Text += String.Format("{0,8}", Iroot.ToString("#####0")) + "  "
                For Idofn = 1 To NDOFN
                    Text     +=     String.Format("{0,13}",     SpecDisp(Iroot,     Ipoin,
Idofn).ToString("0.00000E+00"))
                Next Idofn
                File1.WriteLine(Text)
            Next Iroot
        Next Ipoin

        If IfCQC Then
            Text1 = "Complete Quadratic Combinations (C.Q.C.)"
```

```
            Text2 = "C.Q.C."
        Else
            Text1 = "Square Root of Sum of Squares combinations (S.R.S.S.)"
            Text2 = "S.R.S.S."
        End If
        Text = vbCrLf : Text += vbCrLf
        Text += "============================================================" + vbCrLf
        Text += "          COMBINATIONS OF RESPONSE SPECTRUM RESULTS" + vbCrLf
        Text += "with " + Text1 + vbCrLf
        Text += "============================================================" '+ vbCrLf
        File1.WriteLine(Text)

        Text = vbCrLf
        Text += "***  " + Text2 + " D I S P L A C E M E N T S  ***" + vbCrLf
        Text += vbCrLf
        Text += " NODE       X-displ.       Y-displ.       Z-displ.      XX-rotat.      YY-
rotat.       ZZ-rotat."
        File1.WriteLine(Text)

        For Ipoin = 1 To Npoin
            Text = String.Format("{0,5}", Ipoin.ToString("####0")) + " "
            For Idofn = 1 To NDOFN
                Text       +=       String.Format("{0,15}",       Displ(0,       Ipoin,
Idofn).ToString("0.0000E+00"))
            Next Idofn
            File1.WriteLine(Text)
        Next Ipoin

    End Sub
```

```
    Function ModalAccelerationCalc(Imode As Integer)

        Dim DeltaX, Pend As Single

        ModalAccelerationCalc = Spectrum(1, 2) * Gaccel

        For Ipoin = 2 To NpoiSpec
            If Spectrum(Ipoin, 1) > Period(Imode) Then
                'interpolate the acceleration
                DeltaX = Period(Imode) - Spectrum(Ipoin - 1, 1)
                Pend = (Spectrum(Ipoin, 2) - Spectrum(Ipoin - 1, 2)) / (Spectrum(Ipoin,
1) - Spectrum(Ipoin - 1, 1))
                ModalAccelerationCalc = (Spectrum(Ipoin - 1, 2) + Pend * DeltaX) * Gaccel
                Exit Function
            End If
        Next Ipoin

    End Function
```

```
    Sub WriteSpectrum(File1 As StreamWriter)

        Dim Text1 As String
        Dim Nrows, Icoun As Integer

        Nrows = Int(NpoiSpec / 5)

        Text1 = "Response Design Spectrum (acceleration vs. period)" + vbCrLf
        Text1 += "Elastic accelerations are divided by the behaviour Factor (q = "
        Text1 += String.Format("{0,4}", BehFact.ToString("0.00")) + ")" + vbCrLf
        Text1  +=  "------------------------------------------------------------------------
--------------------------------------" + vbCrLf
        Text1  +=  "    period       ag/g       period       ag/g       period       ag/g
period      ag/g      period      ag/g " + vbCrLf
        Text1  +=  "------------------------------------------------------------------------
---------------------------------------" + vbCrLf

        Icoun = 0
```

```
        For Ipoin = 1 To Nrows
            For Icoup = 1 To 5
                Icoun += 1
                Text1         +=         String.Format("{0,11}",         Spectrum(Icoun,
1).ToString("0.0000E+00"))
                Text1         +=         String.Format("{0,11}",         Spectrum(Icoun,
2).ToString("0.0000E+00"))
            Next Icoup
            Text1 += vbCrLf
        Next Ipoin

        For Icou1 = Icoun + 1 To NpoiSpec
            Text1 += String.Format("{0,11}", Spectrum(Icoun, 1).ToString("0.0000E+00"))
            Text1 += String.Format("{0,11}", Spectrum(Icoun, 2).ToString("0.0000E+00"))
        Next Icou1
        Text1 += vbCrLf

        File1.WriteLine(Text1)

    End Sub
```

```
    Sub SeismicForcesCalculationCQC()
        'calculation of seismic forces with Forces Method

        Dim Disp As Double

        Dim  SeismicForce(NROOT,  Npoin,  NDIME),  ShearForces(NROOT,  Npoin,  NDIME),
BaseShearForce(NROOT, NDIME) As Double
        ReDim ModalAcc(NROOT)
        ReDim SeismicForceCQC(Npoin, NDIME)
        ReDim ShearForcesCQC(Npoin, NDIME)

        For Imode = 1 To NROOT
            ModalAcc(Imode) = ModalAccelerationCalc(Imode)
            For Ipoin = 1 To Npoin
                For Idime = 1 To NDIME
                    Disp = Displ(Ncase + NCOMB + Imode, Ipoin, Idime)
                    SeismicForce(Imode,  Ipoin,  Idime)  =  NodalMass(Ipoin)  *  Disp  *
PartFact(Imode, Idime) * ModalAcc(Imode)
                    BaseShearForce(Imode, Idime) += SeismicForce(Imode, Ipoin, Idime)
                Next Idime
            Next Ipoin
        Next Imode

        'transform seismic forces in storey shears
        For Imode = 1 To NROOT
            For Idime = 1 To NDIME
                ShearForces(Imode, Npoin, Idime) = SeismicForce(Imode, Npoin, Idime)
                For Ipoin = Npoin - 1 To 1 Step -1
                    ShearForces(Imode,  Ipoin,  Idime)  =  ShearForces(Imode,  Ipoin  +  1,
Idime) + SeismicForce(Imode, Ipoin, Idime)
                Next Ipoin
            Next Idime
        Next Imode

        'complete quadratic combinations C.Q.C. of shear forces (D.M. 14.01.2018)
        For Jmode = 1 To NROOT
            For Imode = 1 To NROOT
                For Ipoin = 1 To Npoin
                    ShearForcesCQC(Ipoin,  1)  += Roij(Imode,  Jmode)  *  ShearForces(Imode,
Ipoin, 1) * ShearForces(Jmode, Ipoin, 1)
                    ShearForcesCQC(Ipoin,  2)  += Roij(Imode,  Jmode)  *  ShearForces(Imode,
Ipoin, 2) * ShearForces(Jmode, Ipoin, 2)
                Next Ipoin
            Next Imode
        Next Jmode

        For Ipoin = 1 To Npoin
            ShearForcesCQC(Ipoin, 1) = Math.Sqrt(ShearForcesCQC(Ipoin, 1))
```

```
            ShearForcesCQC(Ipoin, 2) = Math.Sqrt(ShearForcesCQC(Ipoin, 2))
        Next Ipoin

        'and now, finally, calculate storey forces as differences of storey shears
        For Idime = 1 To NDIME
            SeismicForceCQC(Npoin, Idime) = ShearForcesCQC(Npoin, Idime)
        Next Idime
        For Ipoin = Npoin - 1 To 1 Step -1
            For Idime = 1 To NDIME
                SeismicForceCQC(Ipoin,   Idime)   =   ShearForcesCQC(Ipoin,   Idime)   -
ShearForcesCQC(Ipoin + 1, Idime)
            Next Idime
        Next Ipoin

    End Sub
```

```
    Sub STRBER(Icase As Integer, Nele1 As Integer, Nele2 As Integer, FileWork3 As String)

        'STRESS CALCULATION FOR BEAM OR WINKLER ELEMENTS
        'in the case of Response Spectrum Analysis

        Dim Index, Idofn, Idof1, Ldofn As Integer
        Dim GlobDisp(), LocDisp(), LocLoa(), Force(,), Sum, Stres(NEVAB) As Double
        Dim Toler As Double = 0.0000000001

        Using sr As StreamReader = File.OpenText(FileWork3) 'stiffness LOCAL matrices file

            ' *** LOOP OVER ELEMENTS

            For Ielem = Nele1 To Nele2

                'read stiffness matrix
                For Icolu = 1 To 78
                    Stiff(Icolu) = sr.ReadLine
                Next Icolu

                ReDim Force(NEVAB, NROOT)

                For Iroot = 1 To NROOT
                    'calculate displacements LocDisp() and NOT equivalent forces LocLoa()
in local coordinates
                    'transformation T matrices are calculated only once after reading data
                    ReDim GlobDisp(NEVAB), LocDisp(NEVAB), LocLoa(NEVAB)

                    For Idofn = 1 To NDOFN
                        Idof1 = Idofn + NDOFN
                        GlobDisp(Idofn) = SpecDisp(Iroot, Inc1(Ielem), Idofn)
                        GlobDisp(Idof1) = SpecDisp(Iroot, Inc2(Ielem), Idofn)
                    Next Idofn

                    For Idofn = 1 To NDOFN
                        Idof1 = Idofn + NDOFN
                        For Jdofn = 1 To NDOFN
                            Ldofn = Jdofn + NDOFN
                            LocDisp(Idofn) += Tmat(Ielem, Idofn, Jdofn) * GlobDisp(Jdofn)
                            LocDisp(Idof1) += Tmat(Ielem, Idofn, Jdofn) * GlobDisp(Ldofn)
                        Next Jdofn
                    Next Idofn

                    'calculate forces in local system: [K] u = f
                    For Ievab = 1 To NEVAB
                        Sum = 0
                        For Jevab = 1 To NEVAB
                            Index = Kpos(Ievab, Jevab)
                            Sum += Stiff(Index) * LocDisp(Jevab)
                        Next Jevab
                        Force(Ievab, Iroot) = Sum
                    Next Ievab
                Next Iroot
```

```
                ReDim Stres(NEVAB)
                For Ievab = 1 To NEVAB
                    If IfCQC Then
                        'Complete Quadratic Combinations (C.Q.C.) of displacements
                        For Jmode = 1 To NROOT
                            For Imode = 1 To NROOT
                                Stres(Ievab) += Roij(Imode, Jmode) * Force(Ievab, Imode)
* Force(Ievab, Jmode)
                                'Stres(Ievab)      +=      Roij(Imode,      Jmode)      *
Math.Abs(Force(Ievab, Imode)) * Math.Abs(Force(Ievab, Jmode))
                            Next Imode
                        Next Jmode
                    Else
                        'Square  Root  of  Sum  of  Squares  (S.R.S.S.)  combinations  of
displacements
                        For Iroot = 1 To NROOT
                            Stres(Ievab) += Force(Ievab, Iroot) ^ 2
                        Next Iroot
                    End If
                    Stres(Ievab) = Math.Sqrt(Stres(Ievab))
                Next Ievab

                For Ievab = 1 To NEVAB
                    If Math.Abs(Stres(Ievab)) < Toler Then Stres(Ievab) = 0.0
                Next Ievab

                'store values in Stre1(,,), Stre2(,,) matrices
                For Idofn = 1 To NDOFN
                    Stre1(Icase, Ielem, Idofn) = -Stres(Idofn)
                    Stre2(Icase, Ielem, Idofn) = Stres(Idofn + NDOFN)
                Next Idofn
            Next Ielem
        End Using

    End Sub
```

```
    Sub RoijCalc()
        'calculation of C.Q.C. Roij correlation coefficients

        Dim Betaij As Double

        ReDim Roij(NROOT, NROOT)

        For Imode = 1 To NROOT
            For Jmode = 1 To NROOT
                'Betaij = Period(Jmode) / Period(Imode)
                Betaij = CircFreq(Imode) / CircFreq(Jmode)
                Roij(Imode, Jmode) = 8 * DampCsi ^ 2 * Betaij ^ 1.5
                Roij(Imode, Jmode) /= (1 + Betaij)
                Roij(Imode, Jmode) /= ((1 - Betaij) ^ 2 + 4 * DampCsi ^ 2 * Betaij)
            Next Jmode
        Next Imode

    End Sub
```