Modal Analysis

Code_Aster, Salome-Meca course material

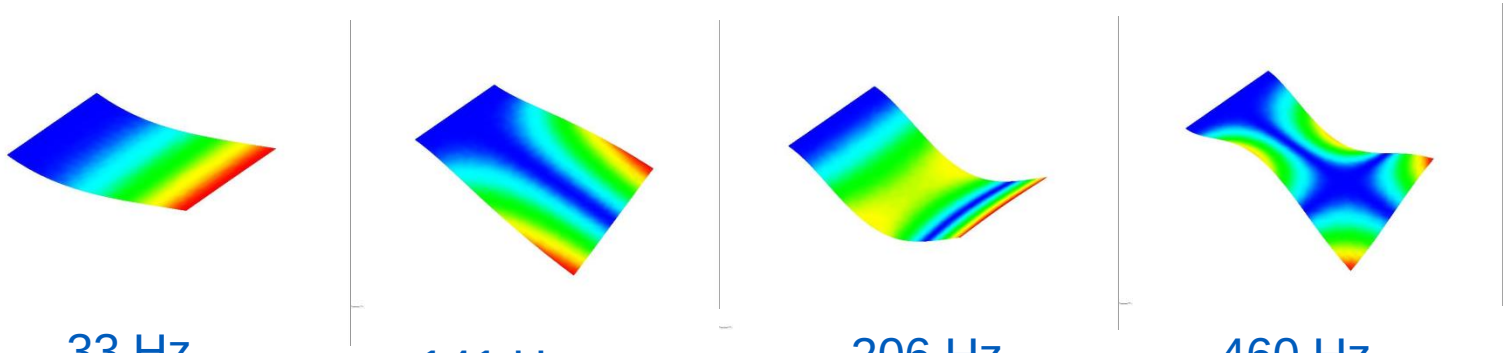
EDF

Outline

- What are natural frequencies & normal modes ?
- Presentation of the eigenvalue problem
- Resolution methods
- Implementation in Code_Aster
- Post-processing & verification
- Tips

What are natural frequencies & normal modes ?

- They represent motions where the interchange between the 2 forms of energy (kinetic & potential) can easily occur
- Mathematically : $(\phi_k, \omega_k) \mid [-\omega_k{}^2 M + K]\,\phi_k = 0 \quad ; \phi_k \neq 0$
- Single DOF : $[-\omega_o{}^2 m + k]\,x_0 = 0 \quad ; x_0 \neq 0 \quad \Rightarrow \quad \omega_o = \sqrt{\frac{k}{m}}$
- They depend on the boundary conditions but not on the external loading
  - Natural frequency (or Eigen frequency) : number of oscillations per second
  - Normal mode (or Eigen vector) : corresponding deformation shape



33 Hz | 141 Hz | 206 Hz | 460 Hz

The eigenvalue problem

- Two types of problems
  - Generalised eigenvalue problem – *for undamped systems*

$$M\ddot{u} + Ku = 0 \quad \Rightarrow \quad (K - \lambda M)\,\Phi = 0$$

  - Quadratic eigenvalue problem – *for damped systems*

$$M\ddot{u} + C\dot{u} + Ku = 0 \quad \Rightarrow \quad \underbrace{(K + \lambda C + \lambda^2 M)\,\Phi = 0}_{\textit{linearization}}$$

$$\underbrace{\left(\begin{bmatrix} C & K \\ K & 0 \end{bmatrix}}_{\mathbf{K}_{quad}} + \lambda \underbrace{\begin{bmatrix} M & 0 \\ 0 & -K \end{bmatrix}\right)}_{\mathbf{M}_{quad}} \begin{bmatrix} \lambda\Phi \\ \Phi \end{bmatrix} = \mathbf{0}$$

- The unknowns
  - Eigen values (pulsations) $0 \le \lambda_1 \le \lambda_2 \le \ldots \le \lambda_n \quad (\lambda = \omega^2)$
  - Eigen vectors (modes) $\Phi_1, \Phi_2, \Phi_3, \ldots, \Phi_n$

Resolution methods – command CALC_MODES

- Modal calculation methods can be classified in 3 categories :
  - Power iteration methods
    - Scope : calculation of extreme values of the spectrum
    - Pros : simplicity, good estimation of the eigenvector in a few iterations
    - Cons : convergence problems, poor identification of multiple modes
    - *Code_Aster* : `OPTION = 'PROCHE'/'SEPARE'/'AJUSTE'`
  - Subspace iteration methods
    - Scope : calculation of a part of the spectrum
    - Pros : reduced size of the problem, less memory required
    - Cons : convergence problems, costly pre- and post- processing steps
    - *Code_Aster* : `OPTION = 'PLUS_PETITE'/'BANDE'/'CENTER'/'TOUT'`

```
SOLVEUR_MODAL = _F( METHODE = 'SORENSEN'/'TRI_DIAG'/'JACOBI' … )
```

  - QR/QZ type methods
    - Scope : calculation of the whole spectrum (+ optional filtering of the results)
    - Pros : robustness
    - Cons : slow convergence, memory and CPU costs (maximum 10^3 DOFs)
    - *Code_Aster* : `OPTION = 'PLUS_PETITE'/'BANDE'/'CENTER'/'TOUT'`

```
SOLVEUR_MODAL = _F( METHODE = 'QZ' …)
```

Power iteration methods : algorithm illustration

- Standard eigenvalue problem $[A]\cdot x = \lambda \cdot x$
- Determine the largest eigenvalue λ_1

`OPTION_INV = 'DIRECT'`

Algorithm

arbitrary initial vector v_0

for i = 1, ...

$$w = [A]\cdot v_{i-1}$$

$$\lambda^i = \frac{\|[A]\cdot w\|}{\|w\|}$$

$$v_i = \frac{w}{\lambda^i} \text{ if } \lambda^i \neq 0$$

$$\ldots (\lambda^i, v_i) \rightarrow (\lambda_1, u_1)$$

Proof

$(\lambda_j, u_j)_{j=1,n}$ eigenmodes and eigenvalue s

$$|\lambda_1| > |\lambda_2| > |\lambda_3| > \ldots$$

$$v_0 = \sum_j (v_0, u_j) u_j$$

$$\frac{1}{\lambda_1^k} A^k v_0 = (v_0, u_1) u_1 + \sum_{j=2,n} \left(\frac{\lambda_j}{\lambda_1}\right)^k (v_0, u_j) u_j$$

$\neq 0$ if possible

Convergence rate

Power iteration methods : different options

- Smallest eigenvalue $[A]^{-1}$
- Eigenvalue closest to σ $[A-\sigma \cdot I]^{-1}$ → Shift and Invert
- Rayleigh quotient $r(x)=\frac{x^T Ax}{x^T x}$: if x is an eigen vect or $\Rightarrow r(x)=\lambda$

Algorithm

initial normalized vector v_0

for i = 1, ...

$\sigma_i = r(v_{i-1})$

solve for w : $[A - \sigma_i I] \cdot w = v_{i-1}$

$v_i = \frac{w}{\|w\|}$

```
OPTION_INV = 'RAYLEIGH'
```

Subspace iteration methods

- Extrapolation of the iteration method on one eigenvalue to a subspace => methods able to compute many eigenvalues at once
  - More efficient than iterations on one value
- Many “flavours”
  - Lanczos TRI_DIAG
  - Bathe and Wilson JACOBI
  - IRAM – Sorensen SORENSEN
  - …
- Always use **METHODE = 'SORENSEN'** with default options in the command **CALC_MODES** as your first attempt
  - Robust
  - Efficient

Verifications

- Convergence of each method
- *A posteriori* error norm
  on the eigen-vectors/values

Eigenvalue problem

$$[\mathrm{K} - \omega^2 \cdot \mathrm{M}] \cdot x = 0$$

$$error_i = \left\| [\mathrm{K}] \cdot v_i - \omega_i^{\,2} [\mathrm{M}] \cdot v_i \right\|$$

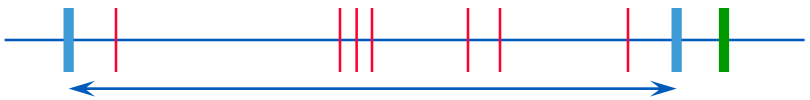
if $error_i \leq threshold$

OK

else

< F > atal error

- Sturm verification : eigenvalue count in a given range



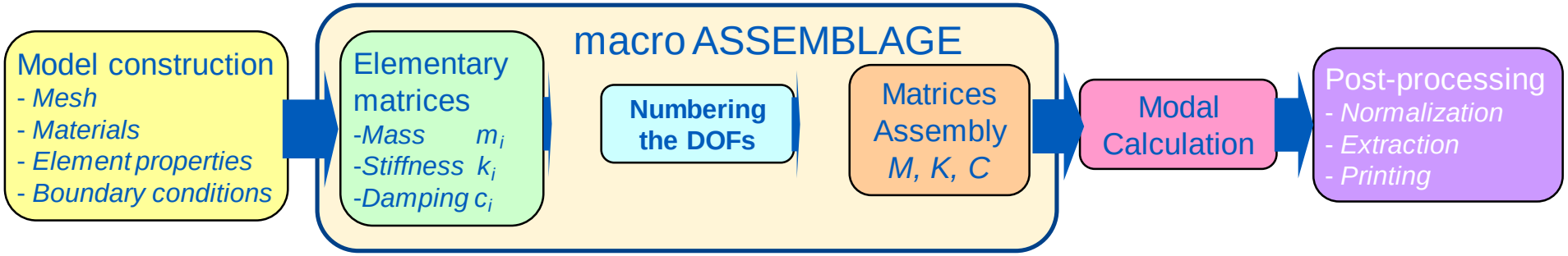
frequency range for post-verification

Frequencies requested by the user

Frequencies calculated but not requested by the user

Implementation in *Code_Aster*

- Main steps for carrying out a modal analysis



- Code_Aster solving commands for modal calculation
  - **CALC_MODES** uses subspace and inverse methods defined according to the **OPTION =** and **SOLVEUR_MODAL=_F(METHODE =)** keywords
  - ***MODE_ITER_CYCL*** *modal calculation of a structure with cyclic symmetry*

Subspace methods in *Code_Aster* : CALC_MODES

Arnoldi IRAM – Sorensen
Bathe and Wilson
Lanczos

- Three subspace iterative methods are available
  - Same principle as power iteration methods but formulated in subspaces (several frequencies are calculated at a time)
  - Best efficiency is obtained with the Arnoldi (IRAM – Sorensen) method

- Recommended method
  - **SORENSEN**
    - Fast and low CPU cost
    - Robust
    - Calculates multiple modes (same frequency but different/orthogonal modes)
    - Can obtain rigid body modes (unconstraint movements)
      - (**OPTION = 'BANDE'** with lower bound 0. or slightly negative)

CALC_MODES command options

- Calculate the `nbf` lowest frequencies
  - **OPTION = 'PLUS_PETITE'**
  - **NB_FREQ = nbf**
- Calculate all frequencies in a given range `[f_min, f_max]`
  - **OPTION = 'BANDE'**
  - **FREQ = [f_min, f_max]**
- Calculate all frequencies around a given reference `f_ref`
  - **OPTION = 'CENTER'**
  - **FREQ = f_ref**
- 3 methods
  - **'JACOBI'** Bathe and Wilson method
  - **'TRI_DIAG'** Lanczos method
  - **'SORENSEN'** **evolution of the Arnoldi method**

    preferred method (by default)

CALC_MODES – recommended method SORENSEN

- Syntax

```
mode  = CALC_MODES (MATR_RIGI = K_MATR,
                    MATR_MASS = M_MATR,
                    OPTION    = 'PLUS_PETITE'    / 'BANDE',
                    CALC_FREQ = _F( NMAX_FREQ = n / FREQ = (f1,f2)),
                    SOLVEUR_MODAL = _F( METHODE = 'SORENSEN') )        << default
```

- Many options and parameters

  Normally reserved for advanced users

CALC_MODES – example

- Message output file

```
mode   = CALC_MODES (MATR_RIGI = K_MATR, MATR_MASS = M_MATR,
                          OPTION = 'PLUS_PETITE', CALC_FREQ = _F( NMAX_FREQ = 5))
```

```
 ---------------------------------------------------------------------
  Les fréquences calculées sont comprises entre :
     Fréquence inférieure :  7.02615E+01
     Fréquence supérieure :  4.20512E+02
 ---------------------------------------------------------------------
     Calcul modal : Méthode d'itération simultanée
                    Méthode de Sorensen

 numéro    fréquence (HZ)     norme d'erreur
    1       7.02615E+01       9.87164E-13
    2       1.28256E+02       3.73093E-13
    3       1.40530E+02       3.24591E-13
    4       1.89140E+02       2.11480E-13
    5       2.11116E+02       1.53475E-13

  Norme d'erreur moyenne   :  1.72031E-13 => error norm check (by default error < 1.E-6)
 ---------------------------------------------------------------------

                    Vérification à posteriori des modes
  Dans l'intervalle ( 7.00856E+01 , 4.21562E+02 ) il y a bien 20 fréquence(s). => Mode number check
 ---------------------------------------------------------------------
```

Pre/post-processing command : INFO_MODE

- Count the number of frequencies in (a) given interval(s)

```
f_table = INFO_MODE(MATR_RIGI = K_MATR, MATR_MASS = M_MATR,
                    FREQ = (f1, f2, f3, …) )
```

- TABLE_FREQ = f_table : can be used as input to CALC_MODES
  - *Special interest for processor mapping to frequency intervals in parallel computing*

- Recommended interval size for efficient CALC_MODES
  - 10 to 30 frequencies per interval

High Performance Computing: CALC_MODES/INFO_MODE

Frequency intervals & parallellisation

- Determine the frequency distribution for the desired range by partitioning the range into smaller intervals
  - Saves CPU time and memory and improves algorithm convergence
  - Possible parallel computing of the partitioned range
- Tip : choose intervals of uniform distribution of modes
  - Count the modes in each of the intervals (INFO_MODE)
  - Change the range partitioning as to acquire an *almost* constant number of modes in each interval (balance the loads for multi processing instances)
- Parallelisation technique
  - First level : many frequency intervals
    - CALC_FREQ =_F (FREQ = (f1,f2,f3,…))
    - **Gain in time**/Gain in memory
  - Second level : direct parallel linear solver 'MUMPS'
    - Gain in time / **Gain in memory**

Resulting data structure : storage indices

NUME_ORDRE

- In the **mode_meca** data structure, the modes are sorted by ascending frequency. **NUME_ORDRE** is the mode position in the calculated data structure.

NUME_MODE

- Mode position in the full spectrum

NUME_ORDRE	NUME_MODE	FREQ
1	23	2.51972E+01
2	24	2.63652E+01
3	25	2.78854E+01
4	26	2.79978E+01
5	27	2.89328E+01

Additional remarks : power iteration methods

`OPTION = 'PROCHE'/'SEPARE'/'AJUSTE'`

- Direct and Rayleigh inverse options
  - Costly algorithms : many factorizations are required
  - `OPTION = 'PROCHE'` does not calculate multiple modes (modal position false)
  - Useful to calculate some eigenvalues
  - Refine the calculation of an eigenvalue (and eigenvector) found by another algorithm

- Verification
  - Error norm on eigenmodes

Resulting data structure : saved parameters

- Generalized mass or stiffness

$$m_x = x^T M x \quad k_x = x^T K x \quad \omega = \frac{k_x}{m_x}$$

$$v(t) = \sum_i \alpha_i(t) x_i$$

$$M\ddot{v} + Kv = f \quad \Rightarrow \quad m_{x_i}\ddot{\alpha}_i + k_{x_i}\alpha_i = f_i$$

- Effective modal mass (unitary)

MASS_EFFE_ (UN_) D *

U_d normalized vector in $\Re^3$

$$m_{x,d} = \frac{\left(x^T M U_d\right)^2}{x^T M x} \quad \text{effective modal mass}$$

$$\sum_i m_{x_i,d} = \text{mass}_{\text{structure}}$$

$$\overline{m}_{x,d} = \frac{1}{\text{mass}_{\text{structure}}} m_{x,d} \quad \text{unit effective modal mass}$$

Participation Factor

FACT_PARTICI_D *

$$p_{x,d} = \frac{\left(x^T M U_d\right)}{x^T M x} \quad \text{participation factor}$$

$$\text{Property: } \sum_i \left(p_{x_i,d}\right)^2 m_{x_i} = \text{mass}_{\text{structure}}$$

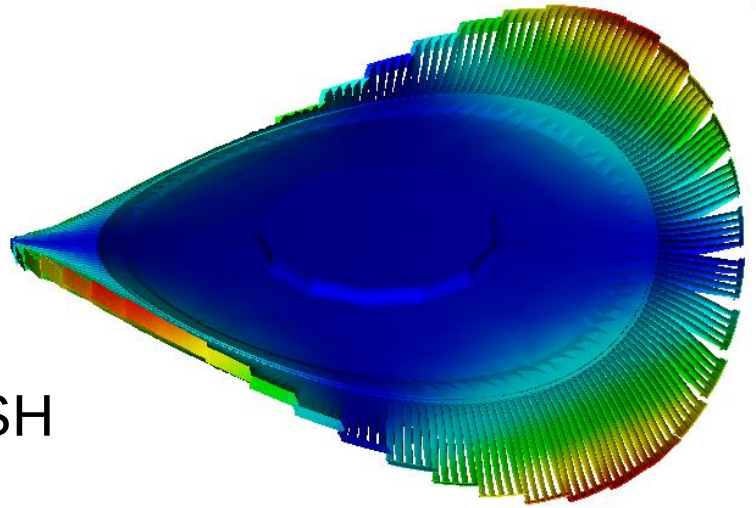
Post-processing command : NORM_MODE

- Normalising
  - By default : largest physical DOF equal to 1
  - **NORME = 'MASS_GENE'** $x^T M\ x = 1$
  - **NORME = 'RIGI_GENE'** $x^T K\ x = 1$
  - Largest physical DOF equal to 1 taken within a group of DOF
    - **NORME = 'TRAN'** **(DX,DY,DZ)**
    - **NORME = 'TRAN_ROTA'** **(DX,DY,DZ, DRX,DRY,DRZ)**
    - **NOEUD = no** and **NOM_CMP=DOF** **DOF(no) = 1**
    - **SANS_CMP = (CMP1, ...)** group without these DOFs
    - **AVEC_CMP = (CMP1, ...)** group composed of these DOFs

Post-processing command : EXTR_MODE

- Concatenate **mode_meca** data structures produced by **CALC_MODES**
- Extract and sort the modes by :
  - Elimination
    - **MASS_GENE** < threshold
    - **MASS_EFFE_UN** < threshold
  - Selection
    - **NUME_ORDRE** list
    - **NUME_MODE** list
- Print the total value of **MASS_ ***
- Detection of duplicate modes from several **mode_meca** to be appended

Printing & visualization of normal modes



- Within SALOME_MECA/PARAVIS
  - “Macro/modes”
  - Magnitudes are arbitrary

- Or with other tools : by example GMSH

```
IMPR_RESU(      FORMAT='GMSH', UNITE=37,
                RESU=_F(RESULTAT=modes,
                NOM_CHAM='DEPL',
                TYPE_CHAM='VECT_3D', NOM_CMP=('DX','DY','DZ',),),)
```

- Printing frequencies in the .resu file

```
IMPR_RESU(      RESU=_F(RESULTAT=modes, TOUT_CHAM='NON', NOM_PARA=('FREQ',)))
```

- The normal modes are simple displacement fields

Tips

- Weigh the model (**POST_ELEM**)
- Estimate the number of frequencies (**INFO_MODE**)
- Use the **'BANDE'** option
- Read doc U2.06.01 => practical handbook of modal computation
- Use **parallelisation** if the number of frequencies is significant
- Take into account the error or alarm messages **<F>** or **<A>**
  - Detecting null pivot, shift change
  - Important norm error on a mode
  - Calculated number of frequencies not consistent with the STURM count verification

End of presentation

Is something missing or unclear in this document?

Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about Code_Aster training materials.

Do not hesitate to share with us your comments on the Code_Aster forum dedicated thread.