Modal Analysis

VU 325.100

Florian TOTH

2020S

Free Oscillation Eigenvalue Problem

Linear, Single DoF Oscillator

$$m\ddot{x} + c\dot{x} + kx = f(t) \tag{1}$$

- inhomogeneous, linear, second order ODE with constant coefficients
- homogeneous solution describes transient response
- particulate solution describes steady-state response

Complex Representation

A physical signal can be represented by the real part of it's complex representation

$$x(t) = \Re\{\hat{x}e^{\hat{s}t}\} \quad \text{with } \hat{x}, \hat{s} \in \mathbb{C} \tag{2}$$

- complex representation is often more convenient to work with
- amplitude and phase encoded in $\hat{x} = a + bj$ with $a, b \in \mathbb{R}$
- damping and frequency encoded in $\hat{s}$

Derive the relations for amplitude, phase, damping and frequency for the real valued signal $x(t) = x_0 e^{\sigma t} \cos(\varphi + \omega t)$.

Multi Degree of Freedom System

A system of n coupled, linear, second-order ODEs may be written in matrix form

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{C}\dot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{f}(t) \tag{3}$$

with the vector of degrees of freedom $\boldsymbol{x} = [x_1, \ldots, x_n]^T$,
$n \times n$ mass, damping and stiffness matrices $\boldsymbol{M}$, $\boldsymbol{C}$ and $\boldsymbol{K}$, and
time dependent forcing vector $\boldsymbol{f}(t) = [f_1(t), \ldots, f_n(t)]^T$.

- can be obtained in various ways
- can be recast into first order system using $\boldsymbol{y} = [\boldsymbol{x}, \dot{\boldsymbol{x}}]^T$

Free, un-damped Oscillations

- free oscillations occur without excitation, thus $\boldsymbol{f} = \boldsymbol{0}$
- un-damped implies $\boldsymbol{C} = \boldsymbol{0}$

Inserting into the general system (3) simplifies to

$$\boldsymbol{M}\ddot{\boldsymbol{x}} + \boldsymbol{K}\boldsymbol{x} = \boldsymbol{0} \tag{4}$$

We can insert the general solution of the from

$$\boldsymbol{x}(t) = \Re\{\hat{\boldsymbol{x}} e^{j\omega t}\} \tag{5}$$

to obtain the un-damped, free-oscillation eigenvalue problem

$$[\boldsymbol{K} - \omega^2 \boldsymbol{M}]\hat{\boldsymbol{x}} = 0 \tag{6}$$

Types of Eigenvalue Problems

standard	$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$
generalised	$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{B}\boldsymbol{x}$
quadratic	$[\boldsymbol{K} + \lambda \boldsymbol{C} + \lambda^2 \boldsymbol{M}]\boldsymbol{x} = \boldsymbol{0}$
polynomial	$[\boldsymbol{A}_0 + \lambda \boldsymbol{A}_1 + \cdots + \lambda^n \boldsymbol{A}_n]\boldsymbol{x} = \boldsymbol{0}$
nonlinear	$\boldsymbol{A}(\lambda)\boldsymbol{x} = \boldsymbol{0}$

Types of Eigenvalue Problems

standard	$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x}$
generalised	$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{B}\boldsymbol{x}$
quadratic	$[\boldsymbol{K} + \lambda \boldsymbol{C} + \lambda^2 \boldsymbol{M}]\boldsymbol{x} = \boldsymbol{0}$
polynomial	$[\boldsymbol{A}_0 + \lambda \boldsymbol{A}_1 + \cdots + \lambda^n \boldsymbol{A}_n]\boldsymbol{x} = \boldsymbol{0}$
nonlinear	$\boldsymbol{A}(\lambda)\boldsymbol{x} = \boldsymbol{0}$

Conversion to Standard Form

A generalised EV problem can transformed to a standard one by pre-multiplying $\boldsymbol{B}^{-1}$ or $\boldsymbol{A}^{-1}$ giving

$$\boldsymbol{B}^{-1}\boldsymbol{A}\boldsymbol{x} = \lambda\boldsymbol{x} \quad \text{or} \quad \boldsymbol{A}^{-1}\boldsymbol{B}\boldsymbol{x} = \frac{1}{\lambda}\boldsymbol{x}$$

A quadratic EV problem my can be converted to a generalised one by $\boldsymbol{y} = \lambda\boldsymbol{x}$ and

$$\begin{bmatrix} \boldsymbol{K} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} = \lambda \begin{bmatrix} -\boldsymbol{C} & -\boldsymbol{M} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \end{bmatrix} \tag{7}$$

The transformation in (7) is not the only possible one. Note also the similarity to the transformation of higher order ODEs.

Solving an Eigenvalue Problem

We consider the standard eigenvalue problem

$$\boldsymbol{A}\boldsymbol{x} = \lambda \boldsymbol{x} \quad \text{or equivalently} \quad (\boldsymbol{A} - \lambda \boldsymbol{I})\boldsymbol{x} = \boldsymbol{0} \tag{8}$$

which has a non-trivial solutions ($x_i \neq 0$) if

$$\det(\boldsymbol{A} - \lambda \boldsymbol{I}) = 0 \tag{9}$$

- The aim is to compute the roots λ_i of the characteristic equation (9) of the eigenvalue problem (8).
- The roots are called *Eigenvalues*.
- For each eigenvalue one can compute a corresponding eigenvector $\boldsymbol{x}_i$.

The eigenvectors are not unique as $(\boldsymbol{A} - \lambda_i \boldsymbol{I})$ is singular. Unique eigenvectors are obtained by setting constraints e.g. on their length.

Eigenvalue Solvers

By hand find the roots of the characteristic polynomial.

Vector iteration computes a single eigenvalue/vector iteratively. Also: inverse vector iteration, Rayleigh quotient Iteration, ...

QR-algorithm computes all eigenvalues iteratively using QR-decompositions.

Subspace iterations extract a sub-set of eigenvalues at one end of the spectrum. Important algorithms: Lanczos, Arnoldi.

Contour-integral based algorithms calculate all eigenvalues inside a given contour in the complex plane.

Eigenvalue solvers available in Matlab, numpy/scipy typically use combinations of above techniques. They often use publicly available libraries, e.g. LAPACK[a], ARPACK[b], or FEAST[c].

[a] `http://www.netlib.org/lapack`
[b] `http://www.caam.rice.edu/software/ARPACK/`
[c] `http://www.ecs.umass.edu/~polizzi/feast/index.htm`

Vector Iteration

The basic vector iteration to iteratively solve a standard EV problem (8) is

$$\boldsymbol{y}_k = \boldsymbol{A}\boldsymbol{x}_k \quad \text{and} \quad \boldsymbol{x}_{k+1} = \frac{\boldsymbol{y}_k}{\|\boldsymbol{y}_k\|}$$

where $\|\boldsymbol{y}_k\|$ converges towards the largest eigenvalue and $\boldsymbol{x}_k$ towards the corresponding eigenvector.

- only yields largest (in magnitude) eigenvalue
- converges baldy if $\lambda_n/\lambda_{n-1} \approx 1$, i.e. the second largest EV is almost the same size as the largest

Vector iteration is also known as *power iteration*, *power method*, or *von Mises iteration*.

Inverse Vector Iteration

λ is an eigenvalue of $\boldsymbol{A}$ $\quad\Leftrightarrow\quad$ $\mu = 1/\lambda$ is an eigenvalue of $\boldsymbol{A}^{-1}$

Do vector iteration with $\boldsymbol{A}^{-1}$ to obtain the eigenvalue μ with largest magnitude, which gives by $\lambda = 1/\mu$ the eigenvalue with smallest magnitude of $\boldsymbol{A}$. The corresponding eigenvector is equal.

The iteration rule is, thus

$$\boldsymbol{y}_k = \boldsymbol{A}^{-1}\boldsymbol{x}_k \quad \text{and} \quad \boldsymbol{x}_{k+1} = \frac{\boldsymbol{y}_k}{\|\boldsymbol{y}_k\|},$$

where $\|\boldsymbol{y}_k\|$ converges towards μ.

In each iteration we solve the same linear system $\boldsymbol{A}\boldsymbol{y}_k = \boldsymbol{x}_k$ with different right hand side, which can be done efficiently by pre-computing the LU factorisation of the system matrix $\boldsymbol{A}$.

Has the same limitations as vector iteration.

Shifted (inverse) Vector Iteration

- λ is an eigenvalue of $\boldsymbol{A}$
- $\lambda - \sigma$ is an eigenvalue of $\boldsymbol{A} - \sigma \boldsymbol{I}$
- $\mu = \frac{1}{\lambda - \sigma}$ is an eigenvalue of the inverse $(\boldsymbol{A} - \sigma \boldsymbol{I})^{-1} = \boldsymbol{B}$
- a suitable shift point $\sigma \approx \lambda_i$ makes μ_i the largest EV of $\boldsymbol{B}$

One can do vector iteration with $\boldsymbol{B}$ to compute μ and the corresponding eigenvector, which is also an eigenvector of $\boldsymbol{A}$.

If the shift point exactly equals an eigenvalue of $\boldsymbol{A}$, i.e. $\sigma = \lambda_i$ then $(\boldsymbol{A} - \sigma \boldsymbol{I})$ is singular (= not invertible) by definition $(\boldsymbol{A} - \lambda_i \boldsymbol{I})\boldsymbol{x} = 0$. Thus, inverse vector iteration will not converge.

Choose shift points *close to* eigenvalues, i.e. $\sigma = \lambda_i + \delta$ with $\delta > 0$.

Rayleigh Quotient Iteration

Inverse vector iteration with changing shift point σ, i.e.

$$\boldsymbol{y}_k = \boldsymbol{B}_k \boldsymbol{x}_k \quad \text{and} \quad \boldsymbol{x}_{k+1} = \frac{\boldsymbol{y}_k}{\|\boldsymbol{y}_k\|}$$

with shift point equal to an estimate of the Rayleigh quotient of $\boldsymbol{A}$

$$\sigma_k = \frac{\boldsymbol{x}_k^* \boldsymbol{A} \boldsymbol{x}_k}{\boldsymbol{x}_k^* \boldsymbol{x}_k} \quad \text{thus} \quad \boldsymbol{B}_k = (\boldsymbol{A} - \sigma_k \boldsymbol{I})^{-1}$$

- $\boldsymbol{x}_k$ converging towards eigenvectors
- $\|\boldsymbol{y}_k\|$ converging towards eigenvectors μ_k of $\boldsymbol{B}_k$
- desired eigenvalues $\lambda = \sigma_k + \frac{1}{\mu_k}$

A linear system has to be solved in each iteration, but convergence is much faster.

Higher Eigenvalues

- vector iteration only yields ends of the spectrum
- shift could be used to get arbitrary EVs, but we don't know where to shift

- once some eigenvectors are known we can use vector iteration in a direction orthogonal to these eigenvectors
- converges towards next eigenvalue
- needs re-orthogonalisation in each iteration (due to numerical errors)

Gram-Schmid Process

Define a projection of $\boldsymbol{v}$ onto $\boldsymbol{u}$ as

check dot product → ... ortho

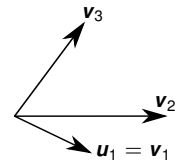
$$p(\boldsymbol{v}, \boldsymbol{u}) = \frac{\boldsymbol{u}^* \boldsymbol{v}}{\boldsymbol{u}^* \boldsymbol{u}} \boldsymbol{u} \tag{10}$$

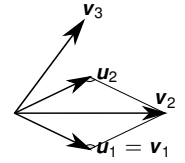
Assume a set of basis vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_k$ then an equivalent orthogonal basis $\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k$ is constructed by

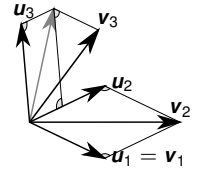
$$\boldsymbol{u}_1 = \boldsymbol{v}_1 \tag{11}$$

$$\boldsymbol{u}_k = \boldsymbol{v}_k - \sum_{j=1}^{k-1} p(\boldsymbol{v}_k, \boldsymbol{u}_j) \tag{12}$$

- $[\boldsymbol{u}_1, \ldots, \boldsymbol{u}_k]$ is an orthogonal matrix
- an ortho*normalized* basis $\boldsymbol{Q} = [\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k]$ is obtained from $\boldsymbol{e}_k = \boldsymbol{u}_k / \|\boldsymbol{u}_k\|$







QR-decomposition

Any $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ may be decomposed such that

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R}$$

where $\boldsymbol{Q}$ is unitary and $\boldsymbol{R}$ is upper triangular.

1. regard the columns of $\boldsymbol{A}$ as the original basis, i.e. $\boldsymbol{A} = [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n]$
2. Compute an orthonormal basis $\boldsymbol{Q}$, e.g. via the Gram-Schmid process[a]
3. Express the $\boldsymbol{v}_k$ via the new basis, i.e. $\boldsymbol{a}_i = \sum_{k=1}^{n} \boldsymbol{e}_k^* \boldsymbol{a}_i \boldsymbol{e}_k$, which can be written as

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{a}_1 & \boldsymbol{a}_2 & \ldots & \boldsymbol{a}_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \ldots & \boldsymbol{e}_n \end{bmatrix} \begin{bmatrix} \boldsymbol{e}_1^* \boldsymbol{a}_1 & \cdots & \boldsymbol{e}_1^* \boldsymbol{a}_n \\ & \ddots & \vdots \\ 0 & & \boldsymbol{e}_n^* \boldsymbol{a}_n \end{bmatrix} = \boldsymbol{Q}\boldsymbol{R}$$

where $\boldsymbol{R}$ is upper triangular due to the orthogonality used in the creation of $\boldsymbol{Q}$.

[a] Alternative methods are: Gives rotations or Housholder reflections.

QR-algorithm

The QR-algorithm uses the QR-decomposition

$$\boldsymbol{A}_k = \boldsymbol{Q}_k \boldsymbol{R}_k \tag{13}$$

where $\boldsymbol{Q}_k$ is unitary and $\boldsymbol{R}_k$ is upper triangular in the iteration rule

$$\boldsymbol{A}_{k+1} = \boldsymbol{R}_k \boldsymbol{Q}_k = \boldsymbol{Q}_k^* \boldsymbol{A}_k \boldsymbol{Q}_k = \boldsymbol{Q}_k^* \cdots \boldsymbol{Q}_0^* \boldsymbol{A}_0 \boldsymbol{Q}_0 \cdots \boldsymbol{Q}_k. \tag{14}$$

Starting from $\boldsymbol{A}_0 = \boldsymbol{A}$ it (usually) converges to $\boldsymbol{A}_k \to \boldsymbol{R}$ in upper triangular form, i.e. computes the Schur decomposition $\boldsymbol{Q}\boldsymbol{R}\boldsymbol{Q}^* = \boldsymbol{A}$ of the matrix $\boldsymbol{A}$.

- $\boldsymbol{Q} = \prod_k \boldsymbol{Q}_k$ is unitary since all $\boldsymbol{Q}_k$ are unitary, therefore
- $\boldsymbol{A}$ and $\boldsymbol{R}$ are similar and have the same eigenvalues.
- As $\boldsymbol{R}$ is upper triangular, eigenvalues can be read from the diagonal.
- Eigenvectors can be computed by inverse vector iteration with shift.

The QR-algorithm computes all eigenvalues of a matrix simultaneously.

Subspace Iteration

only want the first one

- We only want a subset $p < n$ of eigenvalues of $\boldsymbol{A} \in \mathbb{C}^{n \times n}$
- the λ_p should be computed simultaneously

Compute an orthogonal basis $\boldsymbol{X} \in \mathbb{C}^{n \times p}$, i.e. $\boldsymbol{X}^* \boldsymbol{X} = \boldsymbol{I}$, and use it in the iteration

$$\boldsymbol{Z}_{k+1} = \boldsymbol{A}\boldsymbol{X}_k \quad \text{where} \quad \boldsymbol{X}_k \boldsymbol{R}_k = \boldsymbol{Z}_k$$

is the QR-decomposition of $\boldsymbol{Z}_k$. The largest p eigenvalues appear in the diagonal of $\boldsymbol{R}_k$.

$\boldsymbol{A}\boldsymbol{X}_k = \boldsymbol{X}_k \boldsymbol{R}_k$ is an approximation of a *partial* Schur decomposition.

Convergence Criteria

All iterative methods need a convergence criterion for termination. Usually one can use the relative norm of the update, i.e.

$$\frac{\|\lambda_{k+1} - \lambda_k\|}{\|\lambda_k\|} < \epsilon$$

Criterion
↳ eigenvalue

If the corresponding eigenvector $\boldsymbol{x}_i$ is also calculated, one can take the norm of the residual of the eigenvalue equation, e.g.

$$(\boldsymbol{A} - \lambda_i \boldsymbol{I})\boldsymbol{x}_i = \boldsymbol{r}, \qquad \|\boldsymbol{r}\| < \epsilon$$

norm checking
↳ check if equal to zero

Example for Python function

```
def VectorIteration(A,x0,tolerance=1e-6):
  epsilon = 1.0
  while epsilon > tolerance:
    # compute recursion & update epsilon
  return eigenvalue, eigenvector
```

Exercises

- Templates are available in TUWEL
- Submission of selected tasks in TUWEL
- Solutions to all tasks should be presented and discussed during workshop
- Distribute the work within your team

Exercise Templates: getting started...

Template files are *Jupyter Notebook*s)

1. Setup your Python distribution, e.g. anaconda: `https://www.anaconda.com/download` (use Python 3)
2. Download the templates from TUWEL
3. Open *Jupyter* and load the notebook file

Eigenvalue Solvers

Use a small 4×4 matrix with known eigenvalues to test different eigenvalue solvers.

Tasks

Try out the different methods

1. vector iteration
2. inverse vector iteration
3. Rayleigh quotient iteration
4. higher eigenvalues using inverse vector iteration in orthogonal direction
5. QR-algorithm
6. subspace-iteration
7. numpy/matlab

Tipps

- define functions for the different methods
- check if your implementation works for negative eigenvalues too

Eigenvalues and Mode Shapes

Use the provided mass and stiffness matrix of an FE-model of a rectangular plate. A static problem is solved in the template example to illustrate the problem.

Tasks

1. Visualize the structure of the system matrices (e.g. as image)
2. Visualize the geometry (see template)
3. Compute a different static problem
4. Compute and visualise natural frequencies and oscillation modes for different boundary conditions: free, east edge ($x = x_{\text{min}}$) clamped , all edges clamped

Submission

Each template notebook contains a set of questions, which should be answered during the teamwork-phase. You must submit

- PDF file of the notebook answering the questions, and
- all files necessary to reproduce the results in the notebook.

Submission for Workshop 1

1. Function for vector iteration
2. Compute and plot mode shapes for rectangular plate (free-free)

Submissions are due 2 days before the workshop!

You must submit the PDF file separately (not within an archive) to allow comments on your solution via TUWEL.

The First Team Meeting

1. introduce yourself to your team members
2. appoint chairperson and secretary
3. choose a team name, e.g. the modal hammers
4. agree on the agenda
5. work: discuss the lecture & prepare for the workshop
6. write & post the minutes of the meeting

If everyone agrees you can have more than one meeting per learning cycle.

Tasks for Workshop

With your team

- do the suggested exercises
- summarise your findings in a short presentation (max. 15min)
- prepare for the workshop

The presentation should contain

- your main difficulty
- your main insight
- notes on the tested eigenvalue solvers
- comparison of the plate mode shapes

Distribution of Work

- Work should be shared equally between team members
- Use your individual skills and preferences to assign tasks
- Clearly decide who does what!
- Explain what you did and discuss the results together

Every team member must be able to account for work done by others to a reasonable extent!

Example: *Susan programmed a function for vector iteration, Frank can explain what it does and how to use it.*

Tipps for Efficient Team Learning

- Establish a common means of communication (TUWEL forum, mailing list, WhatsUp group, . . .)
- Share data with your team (file server, github repo, google docs, dropbox, . . .)
- Distribute the work between team members
- Meet twice within a learning circle
  1. discuss the lecture and distribute exercise tasks
  2. discuss your work with the team and prepare for the workshop
- prepare for team meetings

Read the Course Manual!

Dates

all events at Wednesday, 09:00–11:00 in BA 05

04/03/2019 Introduction & course information, overview lecture 1
11/03/2019 First team meeting (attendance for team distribution)
18/03/2019 Team learning
25/03/2019 Workshop 1 (attendance), overview lecture 2
01/04/2019 Team learning
22/04/2019 Workshop 2 (attendance), overview lecture 3
29/04/2019 Team learning
06/05/2019 Workshop 3 (attendance), overview lecture 4
13/05/2019 Team learning
20/05/2019 Team learning
27/05/2019 Workshop 4 (attendance), overview lecture 5
03/06/2019 Team learning
10/06/2019 Team learning
17/06/2019 Workshop 5
24/06/2019 Final test

1. Sign up for the course in TUWEL
2. Form teams of 3-4 students: Teams will be fixed at first team meeting
3. Read the course manual

Definitions I

Two vectors $\boldsymbol{u}$, $\boldsymbol{v}$ are orthogonal if their inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = 0$. For the common vector space $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{C}^n$ the inner product $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ is defined as $\boldsymbol{u}^T \boldsymbol{v}$ and $\boldsymbol{u}^* \boldsymbol{v}$, respectively.

The adjoint matrix $\boldsymbol{A}^*$ of $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ has elements related via $A^*_{ij} = \bar{A}_{ji}$, where the overbar denotes the complex conjugate. It is also termed *conjugate transpose* or *Hermitian transpose*.

A matrix $\boldsymbol{Q} \in \mathbb{C}^{n \times n}$ is unitary if

$$\boldsymbol{Q}^* \boldsymbol{Q} = \boldsymbol{Q} \boldsymbol{Q}^* = \boldsymbol{I},$$

i.e. its adjoint is also its transpose.

Definitions II

A matrix $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ is similar to a matrix $\boldsymbol{C} \in \mathbb{C}^{n \times n}$, $\boldsymbol{A} \sim \boldsymbol{C}$, if and only if there is a non-singular matrix $\boldsymbol{S}$ such that

$$\boldsymbol{S}^{-1}\boldsymbol{A}\boldsymbol{S} = \boldsymbol{C}.$$

Similar matrices have *equal eigenvalues* with equal multiplicities.

If $\boldsymbol{A} \in \mathbb{C}^{n \times n}$ then there is a unitary matrix $\boldsymbol{Q} \in \mathbb{C}^{n \times n}$ such that

$$\boldsymbol{Q}^{*}\boldsymbol{A}\boldsymbol{Q} = \boldsymbol{R},$$

is upper triangular, i.e. $R_{ij} = 0 \, \forall \, i > j$. The matrix $\boldsymbol{R}$ is called the Schur form of $\boldsymbol{A}$. The diagonal entries of $\boldsymbol{R}$ are the eigenvalues of $\boldsymbol{A}$.