0.00.5em

0.0.00.5em

0.0.0.00.5em

0em

BrakeSqueal Documentation

Release 0.1

Ali H. Kadar

August 31, 2014

CONTENTS

Contents:

CHAPTER

ONE

BRAKE

1.1 __init__

Source This module defines the following functions:

```
- BrakeClass:

  Python Parent Class for the BrakeSqueal Project

- print_eigs:

  Prints all the eigenvalues on the terminal (with two floating points).

- print_target_eigs:

  Prints the eigenvalues in the target region on the terminal (when flag is false)
  , in the info file (when flag is true).
```

class `brake.__init__.`**`BrakeClass`**(*input_path*, *output_path*, *info_log_file*, *time_log_file*, *log_level*)

`Member Functions of the BrakeClass:`

- __init__
- createInfoLogger
- createTimeLogger
- displayCount
- displayParametersConsole
- displayParametersLog

`brake.__init__.`**`print_eigs`**(*arg*)

Parameters **arg** – eigenvalues

Returns prints eigenvalues on the terminal

`brake.__init__.`**`print_target_eigs`**(*obj*, *arg*, *flag*)

Parameters

- **obj** – object of the class `BrakeClass`
- **arg** – eigenvalues
- **flag** – 0/1 for output on the terminal/in info file

Returns prints eigenvalues on the terminal when flag = 0 else prints output in the info file.

See Also:

logger brake.initialize.logger

CHAPTER

TWO

INITIALIZE

2.1 logger

Source This module defines the following functions:

```
- return_info_logger:

  Creates and returns a python logger object for information logging

- return_time_logger:

  Creates and returns a python logger object for time logging
```

Note:

```
various logging levels
LEVELS = {'notset':logging.NOTSET, #0 --> numerical value (for no logging)
          'debug': logging.DEBUG, #10 (to capture detailed debug information)
          'info': logging.INFO, #20 (to capture essential information)
          'warning': logging.WARNING, #30
          'error': logging.ERROR, #40
          'critical': logging.CRITICAL} #50
```

`brake.initialize.logger.`**`return_info_logger`**`(`*obj*`)`

Parameters **obj** – object of the class `BrakeClass`

Returns `logger_i` python logger object for information logging

`brake.initialize.logger.`**`return_time_logger`**`(`*obj*`)`

Parameters **obj** – object of the class `BrakeClass`

Returns `logger_t` python logger object for time logging

See Also:

Python Logging

2.2 load

Source This module defines the following functions:

```
- load_matrices:

  This function loads the sparse data matrices in .mat format and adds them to a python list
```

brake.initialize.load.**load_matrices**(*obj*)

Parameters **obj** – object of the class `BrakeClass`

Returns `sparse_list` – a python list of matrices in Compressed Sparse Column format of type ‘<type ‘numpy.float64’>’

Procedure:

```
The sparse_list is obtained by loading the vaious .mat files present in the
data_file_list attribute of the BrakeClass and then appending them into
a python list sparse_list.
```

See Also:

scipy.io.loadmat

2.3 assemble

Source This module defines the following functions:

```
- create_MCK:

  Assembles the various component matrices together(for the given angular frequency
  ``omega``) to form the mass(M), damping(C) and stiffness matrix(K).
```

brake.initialize.assemble.**create_MCK**(*obj*, *sparse_list*, *omega*)

Parameters

- **obj** – object of the class `BrakeClass`
- **sparse_list** – a python list of matrices in Compressed Sparse Column format of type ‘<type ‘numpy.float64’>’,
- **omega** – angular frequency

Returns `M` - Mass Matrix, `C` - Damping Matrix, `K` - Stiffness Matrix

Raises Assemble_BadInputError, When a matrix in the list is not sparse

Raises Assemble_BadInputError, When a matrix in the list is not square

Raises Assemble_BadInputError, When the matrix are not of the same size

Procedure:

```
The M , C , K are assembled as follows:

- M = m
- C = c1+c2*(omega/omegaRef)+c3*(omegaRef/omega)
- K = k1+k2+k3*math.pow((omega/omegaRef),2)
```

See Also:

scipy.sparse.linalg.onenormest

2.4 shift

Source This module defines the following functions:

```
- shift_matrices:

  Transform the qevp using shift and invert spectral transformations.
```

`brake.initialize.shift.`**`shift_matrices`**(*obj*, *m*, *c*, *k*, *tau*)

Parameters

- **obj** – object of the class `BrakeClass`
- **m** – Mass Matrix
- **c** – Damping Matrix
- **k** – Stiffness Matrix
- **tau** – Shift

Returns `M`, `C`, `K` - Shifted Mass, Damping and Stiffness Matrix respectively

Procedure:

```
The M , C , K are obtained as follows:

 M = m
 C = 2 * tau * m + c
 K = tau_squared * m + tau * c + k
```

See Also:

scipy.sparse.linalg.onenormest

2.5 scale

Source This module defines the following functions:

```
- scale_matrices:

  Scales the M, C, K matrices using 2-scalers before linearization.
```

`brake.initialize.scale.`**`scale_matrices`**(*obj*, *m*, *c*, *k*)

Parameters

- **obj** – object of the class `BrakeClass`
- **m** – Mass Matrix
- **c** – Damping Matrix
- **k** – Stiffness Matrix

Returns `M`, `C`, `K` - Shifted Mass, Damping and Stiffness Matrix respectively

Returns scaling parameters, - `gamma` and `delta`

Procedure:

```
The M , C , K, gamma, delta  are obtained as follows:

M = gamma*gamma*delta*m;
C = gamma*delta*c;
K = delta*k;
gamma = math.sqrt(k_norm/m_norm);
delta = 2/(k_norm+c_norm*gamma);
```

Note:

```
scipy.sparse.linalg.onenormest - Computes a lower bound of the 1-norm of a sparse matrix.
In the disk brake modelling theory spectral norm has been used but since I could not
find a better(less computational cost) way to obtain this in python, I have used 1-norm
approximation
```

See Also:

scipy.sparse.linalg.onenormest

2.6 diagscale

Source This module defines the following functions:

```
- normalize_cols:

  Returns a diagonal matrix D such that every column of A*D has eucledian norm = 1.

- norm_rc:

  Returns diagonal matrix DL and DR such that every row and every column of DL*Y*DR
  has euclidean norm ~ 1.

- diag_scale_matrices:

  Diagonally scales the shifted scalar-scaled matrices using DL, DR to improve the
  condition number.
```

brake.initialize.diagscale.**diag_scale_matrices**(*obj*, *M*, *C*, *K*)

Parameters

- **obj** – object of the class `BrakeClass`
- **M** – Mass Matrix
- **C** – Damping Matrix
- **K** – Stiffness Matrix

Returns `M`, `C`, `K` - Diagonally Scaled Mass, Damping and Stiffness Matrix respectively

Returns `DR` - Matrix that normalize the columns

Procedure:

```
The M , C , K, DR  are obtained as follows:

M = DL * M * DR
C = DL * C * DR
K = DL * K * DR
```

`brake.initialize.diagscale.`**`norm_rc`**`(`*Y*`)`

Parameters **Y** – – matrix that needs to be normalized across both columns and rows

Returns `DL` - `DL` = ... `Drow3` * `Drow2` * `Drow1` * `I`

Returns `DR` - `DR` = `I` * `Dcol1` * `Dcol2` * `Dcol3`......

Procedure:

```
The DL and DR are obtained as a converging sequence :

set Dcol = normalize columns(Y)
set DR = DR * Dcol
set Y = Y * Dcol
set Drow = normalize rows(Y) = normalize columns(Y.T)
set DL = Drow * DL
set Y = Drow * Y
when Dcol and Drow are sufficiently close to I or max no of iterations reached
STOP and return, else continue with step 1.
```

`brake.initialize.diagscale.`**`normalize_cols`**`(`*A*`)`

Parameters **A** – – matrix that needs to be normalized(columnwise)

Returns `D` - diagonal matrix such that every column of `A*D` has eucledian norm = 1.

Procedure:

```
D is obtained as follows:
square the elements of A and sum each column
set the diagonal elemnts of D as the inverse square root of the column sums
```

See Also:

scipy.sparse.linalg.eigs Documentation of the Python eigs command

2.7 unlinearize

Source This module defines the following functions:

```
- unlinearize_matrices:

  To obtain the eigenvectors prior linearization of the QEVP from the
  resulting eigenvectors (after classical companion linearization of the QEVP
  to obtain the generalized eigenvalue propblem).
```

`brake.initialize.unlinearize.`**`unlinearize_matrices`**`(`*evec*`)`

Parameters **evec** – eigenvectors of the generalized eigenvalue propblem

Returns `evec_prior` - eigenvectors prior linearization of the QEVP

Procedure:

```
The evec_prior are obtained as follows:

check for every vector i of the GEVP(evec) (consider MATLAB convention)
if norm(evec[1:n,i]) > norm(evec[n+1:2*n,i]) set evec_prior[:,i] = evec[1:n,i]
else set evec_prior[:,i] = evec[n+1:2*n,i]
```

See Also:

diagscale brake.initialize.diagscale

CHAPTER

THREE

SOLVE

3.1 projection

Source This module defines the following functions:

```
- obtain_projection_matrix:

 This function forms the Projection Matrix by solving the quadratic eigenvalue problem
 for each base angular frequency.
```

brake.solve.projection.**obtain_projection_matrix**(*obj*)

Parameters **obj** – object of the class `BrakeClass`

Returns `Q` - projection matrix

Procedure:

```
The projection matrix is obtained as follows:

- Obtain the measurment matrix X = [X_real X_imag], with X_real as a list of
  real parts of eigenvectors and X_imag as a list of imaginary parts of
  eigenvectors, corresponding to each base angular frequency in omega_basis.
- Compute the thin svd of the measurment matrix. X = U * s * V
- Set Q = truncated(U), where the truncation is done to take only the significant
  singular values(based on a certain tolerance) into account
```

See Also:

scipy.linalg.svd

3.2 qevp

Source This module defines the following functions:

```
- brake_squeal_qevp:

  For a particuar base angular frequency this function assembles the eigenvalues
  and eigenvectors for different shift points in the target region.

- Obtain_eigs:

  For a particuar base angular frequency and for a particular shift point this
  function evaluates the eigenvalues and eigenvectors
```

brake.solve.qevp.**Obtain_eigs**(*obj*, *freq_i*, *qevp_j*, *omega*, *next_shift*)

Parameters

- **obj** – object of the class `BrakeClass`
- **freq_i** – the index of the base angular freq in
- **qevp_j** – the index of the shift point
- **omega** – ith base angular freq
- **next_shift** – jth shift point in the target region

Returns `la` - eigenvalues, `evec` - eigenvectors

Procedure:

```
The la and evec are obtained as follows:

- load the various component matrices
- assemble the various component matrices together(for the given angular frequency
  omega) to form the mass(M), damping(C) and stiffness matrix(K).
- because we are interested in inner eigenvalues around certain shift points
  next_shift, so we transform the qevp using shift and invert spectral
  transformations.
```

brake.solve.qevp.**brake_squeal_qevp**(*obj*, *freq_i*, *omega*)

Parameters

- **obj** – object of the class `BrakeClass`
- **freq_i** – the index of the base angular freq in
- **omega** – ith base angular freq

Returns `assembled_la` - assembled eigenvalues, `assembled_evec` – assembled eigenvectors

Procedure:

```
The assembled_la and assembled_evec are obtained as follows:

- calculate the next shift point in the target region
- obtain eigenvalues and eigenvectors for that particular shift point
- add the eigenvalues and eigenvectors to assembled_la and assembled_evec respectively
- check if the required area fraction of the target region has been covered. If yes return
  assembled_la and assembled_evec else calculate the next shift point in the target
  region and repeat
```

3.3 solver

Source This module defines the following functions:

```
- qev_sparse:

  Obtains the eigenvalues(smallest in magnitude) and corresponding eigenvectors
  for the given Quadratic Eigenvalue Problem(QEVP) with sparse M, C, K matrices.

- qev_dense
```

```
  Obtains the eigenvalues(smallest in magnitude) and corresponding eigenvectors
  for the given Quadratic Eigenvalue Problem(QEVP) with dense M, C, K matrices.

- gev_sparse

  Obtains the eigenvalues(smallest in magnitude) and corresponding eigenvectors
  for the Generalized Eigenvalue Problem(GEVP) with sparse A, B matrices.

- gev_dense

  Obtains the eigenvalues(smallest in magnitude) and corresponding eigenvectors
  for the Generalized Eigenvalue Problem(GEVP) with dense A, B matrices.
```

Note:

```
The eigs function of PYTHON can calculate the eigenvalues of the generalized eigenvalue
problem A*x=lamda*M*x with the following conditions.
M must represent a real, symmetric matrix if A is real, and must represent a complex,
hermitian matrix if A is complex.
If sigma is None, M has to be positive definite
If sigma is specified, M has to be positive semi-definite

When sigma is specified 'ex 0' then eigs function will calculate the eigenvalues nearest to sigma.
The 'LM' clause along with sigma = 0 can be used to calculate the reciprocal eigenvalues of
Largest Magnitude.
```

`brake.solve.solver.`**`gev_dense`**(*obj*, *A*, *B*)

Parameters

- **obj** – object of the class `BrakeClass`
- **A** – `A x = lamda B x`
- **B** – `A x = lamda B x`

Returns `la` - eigenvalues, `v` - eigenvectors

Raises SOLVER_BadInputError, When the matrix A, B are not all dense

Example:

.

`brake.solve.solver.`**`gev_sparse`**(*obj*, *A*, *B*)

Parameters

- **obj** – object of the class `BrakeClass`
- **A** – `A x = lamda B x`
- **B** – `A x = lamda B x`

Returns `la` - eigenvalues, `v` - eigenvectors

Raises SOLVER_BadInputError, When the matrix A, B are not all in sparse format

Example:

.

`brake.solve.solver.`**`qev_dense`**(*obj*, *M*, *C*, *K*, *no_of_evs*)

Parameters

- **obj** – object of the class `BrakeClass`
- **M** – Mass Matrix
- **C** – Damping Matrix
- **K** – Stiffness Matrix
- **no_of_evs** – No of eigenvalues to be computed

Returns `la` - eigenvalues, `v` - eigenvectors

Raises SOLVER_BadInputError, When the matrix M,C,K are not all dense

Example:

.

`brake.solve.solver.`**`qev_sparse`**(*obj, M, C, K*)

Parameters

- **obj** – object of the class `BrakeClass`
- **M** – Mass Matrix
- **C** – Damping Matrix
- **K** – Stiffness Matrix

Returns `la` - eigenvalues, `v` - eigenvectors

Raises SOLVER_BadInputError, When the matrix M,C,K are not all in sparse format

Example:

.

See Also:

scipy.sparse.linalg.eigs Documentation of the Python eigs command

scipy.sparse.linalg.splu Documentation of the Python splu command

scipy.sparse.linalg.spilu Documentation of the Python spilu command

scipy.linalg.lu_factor Documentation of the Python lu_factor command

scipy.linalg.lu_solve Documentation of the Python lu_solve command

3.4 cover

Source This module defines the following functions:

```
- next_shift

  Implementation of the MonteCarlo Algorithm for choosing the next shift point
  in the target region

- calculate_area_fraction

  Calculates the area fraction covered(of the target rectangle) by the chosen shift points

- draw_circles
```

```
plots a circle corresponding to the next shift point and appends it to the existing plot.
Thus creates a simulation showing how the target region is covered by the various shift points
chosen on fly.
```

`brake.solve.cover.`**`calculate_area_fraction`**(*obj*, *previous_shifts*, *previous_radius*)

Parameters

- **obj** – object of the class `BrakeClass`
- **previous_shifts** – python list for the previous shift points already calculated
- **previous_radius** – corresponding radius of the previous shift points

Returns `area_fraction_covered` - the total area fraction covered with the chosen shift points

Raises `Cover_BadInputError`, When the provided input is not as expected

`brake.solve.cover.`**`draw_circles`**(*obj*, *next_shift*, *next_radius*)

Parameters

- **obj** – object of the class `BrakeClass`
- **next_shift** – the shift point to be shown on the plot
- **next_radius** – the radius of the shift point to be shown

Returns plots a circle corresponding to the next shift point and appends it to the existing plot.

Raises `Cover_BadInputError`, When the provided input is not as expected

`brake.solve.cover.`**`next_shift`**(*obj*, *previous_shifts*=[ ], *previous_radius*=[ ])

Parameters

- **obj** – object of the class `BrakeClass`
- **previous_shifts** – python list for the previous shift points already calculated
- **previous_radius** – corresponding radius of the previous shift points

Returns `next_shift` - next shift point in the target region

Raises `Cover_BadInputError`, When the provided input is not as expected

See Also:

matplotlib.pyplot Documentation of the Python Matplotlib library

randon.py Documentation of the Python random number generator module

CHAPTER

FOUR

ANALYZE

4.1 residual

Source Function definitions for obtaining the residual

See Also:

scipy.sparse.linalg.eigs Documentation of the Python eigs command

scipy.sparse.linalg.spilu Documentation of the Python spilu command

scipy.sparse.linalg.splu Documentation of the Python splu command

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

b

INDEX