vishack

Release 1.0.0

Tsang Terrence Tak Lun

Nov 08, 2021

CONTENTS:

1 Concept **3**
- 1.1 What is VISHack? . . . 3
- 1.2 What is "health check"? . . . 3
- 1.3 Why VISHack? . . . 3
- 1.4 How does VISHack work? . . . 3
- 1.5 How does VISHack identify unhealthy systems? . . . 4

2 Getting Started **5**
- 2.1 Dependencies . . . 5
- 2.2 Installation guide . . . 5
- 2.3 Using VISHack . . . 6

3 Quick Example **7**
- 3.1 Command line . . . 7
- 3.2 Python . . . 7

4 Configuration File **9**
- 4.1 Configuration file description . . . 9
- 4.2 Sample configuration file . . . 10

5 Command Line Utilities **13**
- 5.1 Generate sample configuration file . . . 13
- 5.2 Perform "health check" using a configuration file . . . 13
- 5.3 Read time averaged values from EPICS record . . . 14

6 VISHack Library Reference **15**
- 6.1 vishack.generate_sample_config . . . 15
- 6.2 vishack.HealthCheck . . . 15
- 6.3 vishack.data.ezca . . . 18
- 6.4 vishack.core.evaluate . . . 20
- 6.5 vishack.data.diag . . . 22
- 6.6 vishack.data.diaggui . . . 22
- 6.7 vishack.data.output . . . 25

7 Tutorial **27**
- 7.1 0. Prerequisites . . . 28
- 7.2 1. Diaggui References . . . 28
- 7.3 2. VISHack Configuration file . . . 28
- 7.4 3. Using command line VISHack . . . 29
- 7.5 4. Using VISHack in Python . . . 30
- 7.6 5. Check out the references . . . 31

8 How to Contribute **33**

9 For Developers **35**
9.1 Standards and Tools 35
9.2 Cheat sheet 36

10 Indices and tables **37**

Python Module Index **39**

Index **41**

A self-diagnosis system for vibration isolation systems in KAGRA

VISHack is a python library for system health checks of vibration isolation systems (VIS) in KAGRA.

Main features

- Trigger diaggui measurements and perform system health checks with just one line of command.
- Powered by Python - easy integration with Guardian
- Output system health reports with highlighted alerts.
- Read time average values from EPICS record using Ezca and output them to a file.

Secondary features

- Wraps around `dtt2hdf` - extract frequency series data from diaggui XML files with ease.
- Trigger new measurements using existing diaggui files.

Documentation: https://vishack.readthedocs.io

Repository: https://github.com/gw-vis/vishack

CHAPTER

ONE

CONCEPT

1.1 What is VISHack?

VISHack stands for Vibration Isolation System Health Check.

Here vibration islation systems are referred to suspensions in KAGRA used to suspend optics of the gravitational wave detector.

1.2 What is "health check"?

Health checks tell if a suspension behaves normally. Checks are usually required after traumatic events such as earthquakes. In conventional system health checks, the expert typically uses diaggui measurement templates with reference plots of transfer functions to do the measurement. The new measurements are then compared to the references visually. Then the expert decides, according to the judgement, whether or not the system is healthy.

1.3 Why VISHack?

While the traditional method works fine, VISHack provides a more systematic way to perform consistent system health checks conveniently. Since it is powered by Python, it would be possible to automate system health checks, for example, with Guardian, which is a state-machine software used in KAGRA inherited from LIGO. Nevertheless, VISHack is designed to be extremely easy to use. So, anyone can perform system-diagnosis on vibration isolation systems, say, with a click of a button or a line of command.

1.4 How does VISHack work?

VISHack takes a *Configuration File*, which has specified paths, among other settings, of certain diaggui XML files that are dedicated for system health checks. In the diaggui files, there should be some references of the same measurement that defines the "healthy" state of that particular measurement. VISHack can trigger measurements using the diaggui file, and reads new recordings using the `dtt2hdf` package. VISHack will then compare the measurement results with the references. If the new measurements are very different from the healthy references, then users will be alerted.

1.5 How does VISHack identify unhealthy systems?

VISHack evaluates certain statistical quantities Q using/comparing the new measurement and the healthy references. As of v1.0.0, available options of these quantities are mean squared error (MSE) and maximum absolute error (MAE) between the result and the references, and root mean square (RMS) of the measurement data. There are options to weight/whiten the data with the inverse of the reference before evaluating. These quantities are prefixed with the letter W, i.e. WMSE, WMAE, and WRMS.

The quantities of a particular measurement result are evaluated for each available references and hence the average $\langle Q \rangle$ is obtained. Then, the same evaluation is done using each reference as the measurement and the rest of the available references as the healthy references. Hence, using only the references, the expected value of of the quantity $\langle Q_{\mathrm{ref}} \rangle$ and the standard deviation $\sigma_{Q_{\mathrm{ref}}}$ are obtained.

Finally, the two Q s are compared and presented in units the standard deviation $\sigma_{Q_{\mathrm{ref}}}$, i.e.

$$\langle Q \rangle - \langle Q_{\mathrm{ref}} \rangle = d\, \sigma_{Q_{\mathrm{ref}}},$$

where d is a constant representing the difference between the current measurement and the health references. When $|d|$ is larger than a defined threshold, then the system is considered unhealthy and would require further attention from experts.

CHAPTER

TWO

GETTING STARTED

2.1 Dependencies

2.1.1 Required

- numpy (for basic calculations)
- dtt2hdf (for handling diaggui XML files)

2.1.2 Optional

2.1.3 Planned

- tqdm (for displaying progress bar)

2.2 Installation guide

We recommend using VISHack under Conda environment. Conda is available on k1ctr1 and k1ctr7 in KAGRA.

We will create an environment called *vishack*.

```
conda create -n vishack
```

Then, we can activate it using:

```
conda activate vishack
```

You should see a bracket with the environment name in front of the command prompt.

We proceed to install the required packages. The standard rule for installing packages under a Conda environment is to install everything using `conda install`. If a package is not available, then fallback using `pip install`. In our case, numpy is available in conda-forge while `dtt2hdf` is not. So, we do the following. Notice the order of operation, it matters because the first `conda install` will install an environment specific `pip`.

```
conda install -c conda-forge numpy
```

Then, we confirm that we are using the environment `pip`, not the global `pip` by typing `which pip`. If it shows the global `pip`, then proceed to install pip by `conda install pip`. After confirming that we are using environment specific `pip`, then we can install `dtt2hdf`.

```
pip install dtt2hdf
```

At last, we can install VISHack, by first cloning the repository. Then, change into the downloaded directory and then install it using `pip`

```
git clone https://github.com/gw-vis/vishack.git
cd vishack
pip install .
```

This is it.

To deactivate the environment, simply type

```
conda deactivate
```

2.3 Using VISHack

Check out *Quick Example* for a quick guide and *Tutorial* for a detailed step-by-step guide on how to use VISHack on both command line interface and Python interface. Don't forget to check out *Command Line Utilities* and *VISHack Library Reference* for detailed descriptions of VISHack as well.

CHAPTER

THREE

QUICK EXAMPLE

See here for a quick reference to the essential commands, functions and classes that are needed to use VISHack conveniently. Don't forget to check out *Tutorial* for a detailed step-by-step guide on how to configure and use VISHack in both command line and Python interface.

3.1 Command line

Generate a sample configuration file named *config.ini* for starting:

```
vishack-sample-config --name config.ini
```

After modifying the arguments in *config.ini*, do measurements and checks.

```
vishack --config path/to/config.ini --measure
```

3.2 Python

Similar procedure can also be done in Python interface:

```
from vishack.core.config import generate_sample_config

generate_sample_config(name='config.ini')
```

```
import vishack

hc = vishack.HealthCheck(config='path/to/config.ini')

hc.check(new_measurement=True)
```

CHAPTER

FOUR

CONFIGURATION FILE

Each "health check" of a suspension is defined by a configuration file. The configuration file uses the .ini format and have 7 sections: [General], [Directory settings], [Directories], [Paths], [Coherence], [Transfer function], and [Power spectral density]. The section names are case sensitive so it must be exactly as stated.

4.1 Configuration file description

4.1.1 Section [General]

In the [General] sections, 4 parameters are taken, **Output report**, **Report path**, **Overwrite report**, and **Alert threshold**.

- **Output report** takes a boolean value, true or false. If true, A report will be output to the *Report path* after the diagnosis.
- **Report path** is the path of the report. The report will be in reStructuredText format. Make sure to use .rst extension so it will be recognized in rst viewer or GitHub.
- **Overwrite report** takes a boolean value. If true, then the output report will replace existing files if there's file conflict.
- **Alert threshold** is a float. This defines the standard deviation threshold in which a measurement result is considered to be alarming. We recommend to set this value to 3 as it encloses 99.7% of the cases.

4.1.2 Section [Directory settings]

- **Include subfolders** takes a boolean. If set to true, VISHack will walk through the specified directories and check all diaggui XML files including those in the subfolders, subsubfolders, etc. If set to false, it will only check diaggui XML files in the parent directory.

4.1.3 Section [Directories]

In this section, just list each directory, separated by newline, with the associated diaggui XML files in it.

4.1.4 Section [Paths]

If there are any specific diaggui XML files that are not inside the above directories, specify here.

4.1.5 Section [Coherence]

- **check** takes a boolean. If set to true, VISHack will check coherence plots in the diaggui files.
- **methods** takes a comma-separated list. The list of tests/evaluations to be perform with this "health check". Available tests are MSE, WMSE, MAE, WMAE, RMS, and WRMS.

4.1.6 Section [Transfer function]

- **check** takes a boolean. If set to true, VISHack will check transfer function plots in the diaggui files.
- **methods** takes a comma-separated list. The list of tests/evaluations to be perform with this "health check". Available tests are MSE, WMSE, MAE, WMAE, RMS, and WRMS.

4.1.7 Section [Power spectral density]

- **check** takes a boolean. If set to true, VISHack will check power spectral density plots (actually ASDs in diaggui) in the diaggui files.
- **methods** takes a comma-separated list. The list of tests/evaluations to be perform with this "health check". Available tests are MSE, WMSE, MAE, WMAE, RMS, and WRMS.

4.2 Sample configuration file

VISHack has builtin command line utility function to generate a sample configuration file. Check *Command Line Utilities* to see how to generate a sample configuration file.

The sample configuration file looks like this:

```
[General]
Output report = false
Report path = path/to/report
Overwrite report = false
Alert threshold = 3

[Directory settings]
Include subfolders = false

[Directories]
path/to/bs/xml/directory
path/to/itmy/xml/dirctory

[Paths]
path/to/BS/BS_TM_L.xml
path/to/any/abc.xml

[Coherence]
```

(continues on next page)

(continued from previous page)

```
check = false
methods = MSE,WMSE, MAE, WMAE, RMS, WRMS

[Power spectral density]
check = false
methods = MSE, WMSE, MAE, WMAE, RMS, WRMS

[Transfer function]
check = false
methods = MSE, WMSE, MAE, WMAE, RMS, WRMS
```

Modifiy the entries to suit your needs.

CHAPTER

FIVE

COMMAND LINE UTILITIES

VISHack can be used solely on a command line interface.

If you would like to use VISHack on Python interfaces, see *VISHack Library Reference*.

5.1 Generate sample configuration file

```
$ vishack-sample-config -h
usage: vishack-sample-config [-h] [-n NAME] [-o]

Generate VISHack sample config

optional arguments:
  -h, --help            show this help message and exit
  -n NAME, --name NAME  File name of the config
  -o, --overwrite       Overwrite existing file.
```

Example

We can generate a sample configuration file named sample_config.ini using

```
vishack-sample-config -n sample_config.ini
```

If you would like to overwrite any existing config file that has the same name, just pass in the `-o` or `--overwrite` argument:

```
vishack-sample-config -n sample_config.ini -o
```

5.2 Perform "health check" using a configuration file

```
$ vishack -h
usage: vishack [-h] -c CONFIG [-m]

VISHack suspension health check (self-diagnostic system)

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
```

(continues on next page)

(continued from previous page)

```
                        The path of the .ini config file. If you don't have
                        one, You can generate a smaple config with vishack-
                        sample-config
  -m, --measure         Trigger new measurements
```

Example

To do "health checks" using a configuration file `sample_config.ini`, type

```
vishack -c sample_config.ini
```

In k1ctr workstations, we can use the diaggui XML files to measure new results, in this case we can pass in the `-m` or `--measure` argument:

```
vishack -c sample_config.ini -m
```

This will trigger and save new measurements using the diaggui XML files specified in the configuration file.

5.3 Read time averaged values from EPICS record

```
# vishack-read-time-average -h
usage: vishack-read-time-average [-h] [-c CONFIG] [-g] [-f]

Read time averaged values using EZCA and output to a file.

optional arguments:
  -h, --help            show this help message and exit
  -c CONFIG, --config CONFIG
                        File name of the config
  -g, --get-config      Get a sample configuration file
  -f, --fake-ezca       Use fake ezca instead.
```

CHAPTER

SIX

VISHACK LIBRARY REFERENCE

This is a Python library reference VISHack.

For command line usage, see *Command Line Utilities*.

Main reference

vishack.generate_sample_config([name, overwrite])	Generate a sample health check config
vishack.HealthCheck(config)	Config driven health check class.
vishack.data.ezca	Easy channel access (EZCA) utilities.

6.1 vishack.generate_sample_config

vishack.**generate_sample_config**(*name='sample_config.ini'*, *overwrite=False*)

Generate a sample health check config

Parameters

- **name** (*string, optional*) – Name of the output file.
- **overwrite** (*boolean, optional*) – Overwrite the sample_config.ini in the current directory. If False, a sample config will still be generated. But a number will be appended before the `.ini` extension.

6.2 vishack.HealthCheck

class vishack.**HealthCheck**(*config*)

Bases: `object`

Config driven health check class.

Parameters **config** (*string*) – Path to the config file.

alert

Some alarming results from the report.

Type dict

checklist

A dictionary of tests checklist

Type dict

config
The config parser

Type configparser.ConfigParser

config_path
The path to the config file

Type string

paths
A list of paths to the diaggui XML files to be checked.

Type list of strings

report_header
The report message of the health check

Type string

report
The report of the health check

Type dict

__init__(*config*)
Initiate HealthCheck class with a config file.

Parameters **config** (*string*) – Path to the config file.

Methods

__init__(config)	Initiate HealthCheck class with a config file.
alert_to_string()	Convert alert dictionary to human readable string
check([new_measurement, typelist])	Perform diagnosis checks (health checks).
data_evaluate(data, listof_references, method)	Calculate mean and standard deviation of the evaluations
dict_to_string(dictionary)	Turns a report type dictionary to human readable string (rst)
*evaluate_*(data, reference, method[, df])	Evaluate a statistical quantity between two datasets.
get_alerts([threshold])	Store alerting results from report
print_report(path[, overwrite])	Write health check report to file with human readable format.
reference_evaluate(listof_references, method)	Cross evaluations between references
report_to_string()	Convert health check report dictionary to human readable string

alert_to_string()
Convert alert dictionary to human readable string

Returns **alert_string** – The human readable alert string

Return type string

check(*new_measurement=False*, *typelist=['Transfer function', 'Power spectral density', 'Coherence']*)
Perform diagnosis checks (health checks).

Parameters

- **new_measurement** (*boolean, optional.*) – Trigger new measurement using the diag-

gui XML file. Default False.

- **typelist** (*list of string, optional.*) – The type of checks to be performed. Defaults to check all transfer functions, power spectral density, and coherence in the diaggui XML file.

Returns **report** – The health check report.

Return type dict

Note: Specifying the type of checks here will not override the specification in the configuration file. If you wish to perform a particular type of tests, you must specify in the configuration file as well as specifying here.

---

data_evaluate(*data*, *listof_references*, *method*, *df=1.0*)
Calculate mean and standard deviation of the evaluations

Parameters

- **data** (*array*) – The data to be evaluated
- **listof_references** (*list of arrays*) – A list of references data to compare the data with.
- **method** (*string*) – The type of quantity to be evaluated. Options are 'RMS', 'WRMS', 'MSE', 'WMSE', 'MAE', 'WMAE'.
- **df** (*float, optional*) – The frequency spacing between data points. Default to be 1. Only used when calculating RMS and WRMS.

Returns

- **mean** (*float*) – The mean of all evaluations
- **std** (*float*) – The standard deviation of all evaluations

dict_to_string(*dictionary*)
Turns a report type dictionary to human readable string (rst)

Parameters **dictionary** (*dict*) – The health check report or the alert

Returns **rst_string** – The string in reStructuredText format.

Return type string

evaluate_(*data*, *reference*, *method*, *df=1.0*)
Evaluate a statistical quantity between two datasets.

Parameters

- **data** (*array*) – The data to be evaluated
- **reference** (*array*) – The reference data to be referenced
- **method** (*string*) – The type of quantity to be evaluated. Options are 'RMS', 'WRMS', 'MSE', 'WMSE', 'MAE', 'WMAE'.
- **df** (*float, optional*) – The frequency spacing between data points. Default to be 1. Only used when calculating RMS and WRMS.

get_alerts(*threshold=3*)
Store alerting results from report

Parameters **threshold** (*float, optional.*) – Alert results when the mean of the result is higher than this threshold, which has a unit of sigma. Defaults to 3. (3 sigma encloses 99.7% of the cases)

Returns **alert** – Some alerting results from the health check report.

Return type dict

print_report(*path*, *overwrite=False*)
Write health check report to file with human readable format.

Parameters

- **path** (*string*) – path to the report
- **overwrite** (*boolean, optional*) – Overwrite existing file. If false, path will be renamed before writing the report.

Returns **full_string** – The full string of the report.

Return type string

reference_evaluate(*listof_references*, *method*, *df=1.0*)
Cross evaluations between references

Parameters

- **listof_references** (*list of arrays*) – A list of references data.
- **method** (*string*) – The type of quantity to be evaluated. Options are 'RMS', 'WRMS', 'MSE', 'WMSE', 'MAE', 'WMAE'.
- **df** (*float, optional*) – The frequency spacing between data points. Default to be 1. Only used when calculating RMS and WRMS.

Returns

- **mean** (*float*) – The mean of all evaluations
- **std** (*float*) – The standard deviation of all evaluations

report_to_string()
Convert health check report dictionary to human readable string

Returns **report_string** – The human readable report string

Return type string

6.3 vishack.data.ezca

Easy channel access (EZCA) utilities.

Functions

parallel_read(ezca, channels)	Read channels values and returns a dict of values in the EPICS record.
parallel_time_average(ezca, channels[, ...])	Read process variables channels and returns a dict of time averages
parallel_time_series(ezca, channels[, ...])	Read channels time-series and returns of dict of time-series.

vishack.data.ezca.**parallel_read**(*ezca*, *channels*)
Read channels values and returns a dict of values in the EPICS record.

Parameters

- **ezca** (*ezca.ezca.Ezca*) – Ezca instance.
- **channels** (*list of str*) – The channels to the read

Returns A dictionary with channel names as the key and time-series as the value.

Return type dict

vishack.data.ezca.**parallel_time_average**(*ezca*, *channels*, *duration=1.0*, *fs=None*)
Read process variables channels and returns a dict of time averages

Parameters

- **ezca** (*ezca.ezca.Ezca*) – Ezca instance.
- **channels** (*list of str*) – The channels to be read.
- **duration** (*float, optional*) – The duration (s). Defaults to 1 second.
- **fs** (*float, optional*) – Sampling frequency (s). Note, the sampling frequency of EPICS slow channels is maximum at 8 Hz. If None, will read as fast as it can. Defaults None.

Returns A dictionary with channel names as the keys and the time average as the value.

Return type dict

vishack.data.ezca.**parallel_time_series**(*ezca*, *channels*, *duration=1.0*, *fs=None*)
Read channels time-series and returns of dict of time-series.

Parameters

- **ezca** (*ezca.ezca.Ezca*) – Ezca instance.
- **channels** (*list of str*) – The channels to be read.
- **duration** (*float, optional*) – The duration (s). Defaults to 1 second.
- **fs** (*float, optional*) – Sampling frequency (s). Note, the sampling frequency of EPICS slow channels is maximum at 8 Hz. If None, will read as fast as it can. Defaults None.

Returns A dictionary with channel names as the keys and time-series as the values.

Return type dict

Detailed reference (for power users only)

vishack.core.evaluate	Evaluate statistical quantities from frequency series.

continues on next page

Table 4 – continued from previous page

vishack.data.diag	Library for interfacing with the diagnostic tools command *diag*.
vishack.data.diaggui	A dtt2hdf wrapper for extracting data from diaggui XML output files
vishack.data.output	Library for handling file outputs.

6.4 vishack.core.evaluate

Evaluate statistical quantities from frequency series.

Functions

mae(data, reference)	Maximum absolute error between the data and the reference
mse(data, reference)	Mean-square-error between the data and reference data.
rms(data[, df])	Calculated the expected root-mean-square value from the data
wmae(data, reference)	Maximum absolute error between the whitened data and the reference
wmse(data, reference)	Mean-square-error between the whitened data and reference data.
wrms(data[, df, whitening])	Whiten the data and calcuate the expected root-mean-square value.

vishack.core.evaluate.**mae**(*data*, *reference*)
Maximum absolute error between the data and the reference

Parameters

- **data** (*array*) – The data to be evaluted
- **reference** (*array*) – The reference data.

Returns The maximum absolute error between the data and the reference

Return type float

vishack.core.evaluate.**mse**(*data*, *reference*)
Mean-square-error between the data and reference data.

Parameters

- **data** (*array*) – The data to be evaluted
- **reference** (*array*) – The reference data.

Returns The mean-square-error between the data and the reference.

Return type float

vishack.core.evaluate.**rms**(*data*, *df=1.0*)
Calculated the expected root-mean-square value from the data

Parameters

- **data** (*array*) – The data to be evaluated

- **df** (*float, optional*) – The frequency spacing between data points. Default to be 1.

Returns The expected RMS.

Return type float

Note: Complex arrays are accepted. Absolute values will be taken after whitening. If the data is an amplitude spectral density, then the output will be the expected RMS. If the data is a transfer function, then the output will be the 2-norm.

---

vishack.core.evaluate.**wmae**(*data*, *reference*)

Maximum absolute error between the whitened data and the reference

Parameters

- **data** (*array*) – The data to be evaluted
- **reference** (*array*) – The reference data.

Returns The maximum absolute error between the data and the reference whitened by the inverse of the reference.

Return type float

vishack.core.evaluate.**wmse**(*data*, *reference*)

Mean-square-error between the whitened data and reference data.

Parameters

- **data** (*array*) – The data to be evaluted
- **reference** (*array*) – The reference data.

Returns The mean-square-error between the data and the reference whitened by the inverse of the reference.

Return type float

vishack.core.evaluate.**wrms**(*data*, *df=1.0*, *whitening=None*)

Whiten the data and calcuate the expected root-mean-square value.

Parameters

- **data** (*array*) – The data to be evaluated.
- **df** (*float, optional*) – The frequency spacing between data points. Default to be 1.
- **whitening** (*array, optional*) – The whitening/weighting function. If None, default to be ones.

Returns The whitened expected RMS.

Return type float

Note: Complex arrays are accepted. Absolute values will be taken after whitening. If the data is an amplitude spectral density, then the output will be the expected RMS. If the data is a transfer function, then the output will be the 2-norm.

6.5 vishack.data.diag

Library for interfacing with the diagnostic tools command *diag*.

Functions

make_script(path, lines[, overwrite])	Create a script and make it executable
run_measurement(path[, saveas, remove_tmp])	Run a measurement set up by a diaggui XML file.

vishack.data.diag.**make_script**(*path*, *lines*, *overwrite=False*)
Create a script and make it executable

Parameters

- **path** (*string*) – The path of the script
- **lines** (*list of strings*) – Scripting commands to be written on the script.
- **overwrite** (*boolean, optional*) – Overwrite if the path exists, if not, the file will be saved as a different name. Defaults to False.

vishack.data.diag.**run_measurement**(*path*, *saveas=None*, *remove_tmp=True*)
Run a measurement set up by a diaggui XML file.

Parameters

- **path** (*string*) – The path of the diaggui XML file
- **saveas** (*string, optional*) – Save the measurement as a different file when finished measurement. Defaults to None. If None, it is same as *path*
- **remove_tmp** (*boolean, optional*) – Remove any temporary files that are used to trigger this measurement. Defaults to True.

6.6 vishack.data.diaggui

A dtt2hdf wrapper for extracting data from diaggui XML output files

Classes

Diaggui(path)	Diaggui class for handling and converting diaggui XML file.

class vishack.data.diaggui.**Diaggui**(*path*)
Bases: `object`

Diaggui class for handling and converting diaggui XML file.

Parameters **path** (*string*) – The path to the diaggui XML output file.

items
The output from `dtt2hdf.read_diaggui(path)`.

Type declarative.bunch.bunch.Bunch

path
The path to the diaggui XML output file.

Type string

coh(*channel_a*, *channel_b*, *datatype='results'*)
Read coherence from diaggui file.

Parameters

- **channel_a** (*string*) – The input channel string.
- **channel_b** (*string*) – The output channel string.

Returns

- **f** (*array*) – The frequency axis of the coherence.
- **cohdata** (*array*) – The coherence between *channel_a* and *channel_b*.

csd(*channel_a*, *channel_b*, *datatype='results'*)
Read cross-PSD from diaggui file.

Parameters

- **channel_a** (*string*) – The input channel string.
- **channel_b** (*string*) – The output channel string.

Returns

- **f** (*array*) – The frequency axis of the cross-power sepctral density.
- **csddata** (*array*) – The cross-power spectral density, in complex numbers.

get_reference(*index*)
Read a reference plot from the diaggui XML file

Parameters **index** (*int*) – The index of the reference plot in the diaggui XML

Returns A dictionary with various useful info about the reference.

Return type dict

Note: The dict is taken from dtt2hdf.read_diaggui().references[index]. Useful keys are 'type_name', 'channelA', 'channelB', 'channelB_inv', 'df', 'FHz', 'xfer', 'PSD', 'CSD', 'coherence'.

Example

Here, reference #0 is the transfer function from BS_TM_L to BS_TM_L.

```
In[0]:
 import vishack.data.diaggui
 dg = vishack.data.diaggui.Diaggui(path='data/BS_TML_exc_20200730a.xml')
 dg.reference_dict(0)
```

```
Out[0]:
 {'gps_second': 1238215044.0078125,
  'subtype_raw': 0,
  'f0': 0.0,
```

(continues on next page)

(continued from previous page)

```
 'df': 0.0078125,
 'BW': 0.0117187,
 'window_raw': 1,
 'window': 'Hanning',
 'avgtype_raw': 0,
 'avgtype': 'Fixed',
 'averages': 1,
 'channelA': 'K1:VIS-BS_TM_LOCK_L_EXC',
 'channelB': array(['K1:VIS-BS_TM_OPLEV_LEN_DIAG'], dtype='<U27'),
 'channelB_inv': {'K1:VIS-BS_TM_OPLEV_LEN_DIAG': 0},
 'subtype': 'transfer function B/A in format (Y)',
 'type_name': 'TF',
 'xfer': array([[ 2.0319992e-03+0.0000000e+00j,  1.5529208e-03+1.4785477e-03j,
         9.0984150e-04+1.8886093e-03j, ...,
         1.1092599e+27-2.3797619e+08j,  5.5111208e-08-2.0454582e-04j,
        -4.4841594e-35-4.2907759e-42j]], dtype=complex64)}
```

get_results(*type_name*)

Return the results of a particular type from the diaggui XML file.

Parameters **type_name** (*string*) – The type of results. 'TF', 'COH', 'CSD', or 'PSD'.

Returns A dictionary with all the results with key being the channelA string.

Return type dict

measure()

Measure new results using the diaggui XML file.

psd(*channel_a*, *datatype='results'*)

Read power spectral density from diaggui file

Parameters **channel_a** (*string*) – The channel name of the PSD to be read.

Returns

- **f** (*array*) – The frequency axis of the PSD.
- **psddata** (*array*) – The power spectral density.

Notes

The PSD in diaggui is actually amplitude spectral density (ASD), not PSD.

tf(*channel_a*, *channel_b*, *datatype='results'*)

Derive transfer function from CSD and PSD from diaggui file.

Parameters

- **channel_a** (*string*) – The input channel string.
- **channel_b** (*string*) – The output channel string.

Returns

- **f** (*array*) – The frequency axis of the transfer function.
- **tfdata** (*array*) – The transfer function, defined by B/A, in complex numbers.

6.7 vishack.data.output

Library for handling file outputs.

Functions

append_to_name(name, ext[, appendix])	Rename the name of the file if it exists
get_name_and_ext(path)	Get name and extension from a path
rename(path[, method])	Rename a file by appending something unique before extension.
rename_method_123(name, ext)	Rename the name of the file by appending bracket enclosed number.
rename_method_utc(name, ext)	Rename the name of the file by appending the current utc time.

vishack.data.output.**append_to_name**(*name*, *ext*, *appendix=''*)
Rename the name of the file if it exists

Parameters

- **name** (*string*) – The name of the file, i.e. everything before the extension
- **ext** (*string*) – The extension of the file.
- **appendix** (*string*) – The appendix to append onto the name

Returns **new_path** – The modified path name.

Return type string

vishack.data.output.**get_name_and_ext**(*path*)
Get name and extension from a path

Parameters **path** (*string*) – The path of the file

Returns

- **name** (*string*) – The name of the file, i.e. everything before the extension
- **ext** (*string*) – The extension of the file.

vishack.data.output.**rename**(*path*, *method='utc'*)
Rename a file by appending something unique before extension.

Parameters

- **path** (*string*) – The path of the file to be outputted. With or without extension. If it exists, the new file will be renamed before writing so not to replace the old one.
- **method** (*string*) – The method to rename. Available methods are 'utc' and '123'.

Returns **new_path** – The modified path name.

Return type string

vishack.data.output.**rename_method_123**(*name*, *ext*)
Rename the name of the file by appending bracket enclosed number.

Parameters

- **name** (*string*) – The name of the file, i.e. everything before the extension

- **ext** (*string*) – The extension of the file.

Returns **new_path** – The modified path name.

Return type string

vishack.data.output.**rename_method_utc**(*name*, *ext*)

Rename the name of the file by appending the current utc time.

Parameters

- **name** (*string*) – The name of the file, i.e. everything before the extension
- **ext** (*string*) – The extension of the file.

Returns **new_path** – The modified path name.

Return type string

CHAPTER

SEVEN

TUTORIAL

Using VISHack to do suspension system diagnosis is a 3-step process.

1. Prepare diaggui XML files with references.
2. Prepare VISHack configuration file.
3. Measure and checks (regular-checks) using VISHack.

The first 2 steps are really just preparations that needed to be done once. Step 3 is the actual diagnosis/health check step. There are two ways that can achieve this, using command line or Python.

Here, we present a step-by-step tutorial with an example on how to perform self-diagnosis, otherwise known as "health check", on KAGRA's vibration isolation systems (VIS). (In fact, VISHack would work on any systems that can be measured using diaggui.)

All example files are avaliable at https://github.com/gw-vis/vishack/blob/master/examples

The example files have the following structure:

```
examples
├── configs
│   └── example_config.ini
├── diaggui
│   └── SRM
│       ├── F0
│       │   ├── SRM_IDAMPL_DAMPL_20200929(1).xml
│       │   └── SRM_IDAMPL_DAMPL_20200929(2).xml
│       ├── IP
│       │   └── SRM_IDAMPL_DAMPL_20200929.xml
│       ├── SRM_IDAMPL_DAMPL_20200929(3).xml
│       └── SRM_IDAMPL_DAMPL_20200929.xml
├── example_vishack_healthcheck.py
└── reports
    ├── another_report.rst
    ├── another_report_with_new_threshold.rst
    └── srm_health_report.rst
```

7.1 0. Prerequisites

We assume you know how to use `diaggui` to take and record measurements as references. If you don't know how to use `diaggui`, but still want to perform measurements, good luck.

Basic knowledge of Python is encouraged. If you don't want to use the Python interface, you can still use the command line interface of VISHack, or even an MEDM or Guardian interface, if someone implemented these in the future.

Of course, we also assume that you have VISHack installed. If not, do check out *Getting Started*.

7.2 1. Diaggui References

The first step of the self-diagnosis procedure is to take references of acceptable/accepted measurements. These measurements are those which defines the vital status of the system. For example, a transfer function. The way VISHack works, is that it compares new measurement results to the references in the diaggui file. To prevent excessive false alarms, we recommend to have no less than 3 references of the same measurement, and, the more, the better. Of course, you can always add the measurements to the references when you decided that it was a false alarm.

In general, a single "health check" of a suspension will associate multiple diaggui XML files with references. It is better to move all diaggui XML files into a centralized directory. Subdirectories can also be used for sorting different measurement files.

We recommend to have separate directories for different suspensions. This is because different suspensions can be checked in parallel.

In our case will use a single dummy diaggui XML file *SRM_IDAMPL_DAMPL_20200929.xml* and duplicate it as if they were separate independent measurements. In the file, there are 14 references in total. 7 PSD measurements of the channel *K1:VIS-SRM_IP_DAMP_L_IN1*, and 7 transfer function measurements from *K1:VIS-SRM_IP_IDAMP_L_OUT* to *K1:VIS-SRM_IP_DAMP_L_IN1*.

The XML files are all populated inside the folder *diaggui/SRM*. There are two subfolders *diaggui/SRM/IP* and *diaggui/SRM/F0*, just for demonstration.

7.3 2. VISHack Configuration file

In the *config* folder, there's a config file named *example_config.ini*. We will be using this configuration for the tutorial.

The configuration is as follows:

```
[General]
Output report = True
  # This will generate a report file.

Report path = reports/srm_health_report.rst
  # This is the path of the report.
  # Use .rst extension to view it with reStructuredText compatible applications
  # like GitHub or Read the docs.

Overwrite report = True
  # This will overwrite if there's file conflict.
  # If true, it will append something before writing.

Alert threshold = 3
```

(continues on next page)

(continued from previous page)

```
  # This alert threshold is in unit of standard deviation.
  # If any of the test results is off by this amount compare to the references,
  # it will be reported.

[Directory settings]
Include subfolders = True
  # This will include all diaggui files in the all subfolders, subsubfolders...
  # etc. In the directories specified below.

[Directories]
# Every diaggui XML files will be checked under these directories.
diaggui/SRM/F0
diaggui/SRM/IP

[Paths]
# Alternatively, we can specify individual files directly.
# Repeated paths will not be checked twice.
diaggui/SRM/SRM_IDAMPL_DAMPL_20200929.xml


# Here are the type of results that we will check.
# Use 'check = True' to enable it or else it will not check.
# Type of tests available are mean-square-error (MSE),
# Maximum absolute error (MAE), and root-mean-square (RMS).
# Tests started with a 'W', such as WMSE, will whiten the data
# with reference before evaluating.
[Coherence]
check = True
methods = MSE,WMSE, MAE, WMAE, RMS, WRMS

[Power spectral density]
check = true
methods = MSE, WMSE, MAE, WMAE, RMS, WRMS

[Transfer function]
check = True
methods = MSE, WMSE, MAE, WMAE, RMS, WRMS
```

7.4 3. Using command line VISHack

Note that the directories and paths specified in *config/example_config.ini* are relative to the *example* directory. Therefore, we must run everything under the *example* directory. If you would like to run the tests everywhere, you must specify the full path of the diaggui XML files or the directories in the configuration file.

To run the tests using the config, simply type

```
vishack -c config/example_config.ini
```

This will generate a report named *srm_health_report.rst* in the *report* directory. Feel free to open it to see how it looks like with your favorite editor. If you view it on GitHub, you will notice something special. If the **Overwrite report** argument is set to false in the config, the UTC time will be appended to the file name before outputting the report.

The above command will not trigger new measurements. If you would like to measure new results, add `-m` to the command:

```
vishack -c config/example_config.ini -m
```

If you type this on your local machine it will probably output shell error because the `diag` command is not installed on your local machine but only on k1ctr workstations. Nevertheless, this will still generate the report using the old reports.

During the measurement, you should see meaningless outputs from the diagnostic command line tools `diag`. In the future, we hope to replace this by a progress bar.

7.5 4. Using VISHack in Python

An example Python script is available at *example_vishack_healthcheck.py*.

This is the script.

```
import vishack

# Initial an HealthCheck instance with the config file
srm_hc = vishack.HealthCheck(config='configs/example_config.ini')

# Call check() method to do health check. Use new_measurement=True to trigger
# New measurements. In this case you will have to wait until the measurements
# finish. Only use this with k1ctr workstations.
srm_hc.check(new_measurement=True)
# If you don't have access to workstations, but you still want to check
# diaggui files in hand, you can still do it:
# Uncomment below.
# srm_hc.check(new_measurement=False)

# The new_measurement argument is False by default so specifying it with False
# is actually redundant.

# Since we have already specify to generate a report in the config file,
# There is no need to generate it. In case we want to, we can use the
# print_report() method
srm_hc.print_report(path='reports/another_report.rst', overwrite=True)

# To overwrite the alerts threshold in the config, we can manually generate
# new alerts:
srm_hc.get_alerts(threshold=1.9)


# If we want to check what which files are associated with the alerts,
# We can simply print:
print(srm_hc.alert.keys())
# Exercise: In your local machine, try changing the threshold to 2.0 and see
# what happens to the alert.

# After getting new alerts, we can print new reports.
srm_hc.print_report(
    path='reports/another_report_with_new_threshold.rst',
```

(continues on next page)

(continued from previous page)

```
    overwrite=True
)
```

7.6 5. Check out the references

Do check out *Command Line Utilities* and *VISHack Library Reference* for detailed descriptions on VISHack.

CHAPTER

EIGHT

HOW TO CONTRIBUTE

Try out the package and file an issue if you find any!

CHAPTER

NINE

FOR DEVELOPERS

Contents

- *For Developers*
  - *Standards and Tools*
  - *Cheat sheet*

9.1 Standards and Tools

Please comply with the following standards/guides as much as possible.

9.1.1 Coding style

- **PEP 8**: https://www.python.org/dev/peps/pep-0008/

9.1.2 CHANGELOG

- **Keep a Changelog**: https://keepachangelog.com/en/1.0.0/

9.1.3 Versioning

- **Semantic Versioning**: https://semver.org/spec/v2.0.0.html

9.1.4 Packaging

- **PyPA**: https://www.pypa.io
- **python-packaging**: https://python-packaging.readthedocs.io

9.1.5 Documentation

- **NumPy docstrings**: https://numpydoc.readthedocs.io/en/latest/format.html
- **Sphinx**: https://www.sphinx-doc.org/
- **Read The Docs**: https://readthedocs.org/
- **Documenting Python Code: A Complete Guide**: https://realpython.com/documenting-python-code/

9.2 Cheat sheet

9.2.1 Sphinx

Generate documentation base, in docs/,

```
sphinx-quickstart
```

Select separate build and source files when prompted.

Preview documentation page with modified source, in docs/

```
make html
```

Open index.html with a browser (if this was set as the first page).

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

V

vishack.core.evaluate, 20
vishack.data.diag, 22
vishack.data.diaggui, 22
vishack.data.ezca, 18
vishack.data.output, 25

INDEX

Symbols

__init__() (*vishack.HealthCheck method*), 16

A

alert (*vishack.HealthCheck attribute*), 15
alert_to_string() (*vishack.HealthCheck method*), 16
append_to_name() (*in module vishack.data.output*), 25

C

check() (*vishack.HealthCheck method*), 16
checklist (*vishack.HealthCheck attribute*), 15
coh() (*vishack.data.diaggui.Diaggui method*), 23
config (*vishack.HealthCheck attribute*), 15
config_path (*vishack.HealthCheck attribute*), 16
csd() (*vishack.data.diaggui.Diaggui method*), 23

D

data_evaluate() (*vishack.HealthCheck method*), 17
Diaggui (*class in vishack.data.diaggui*), 22
dict_to_string() (*vishack.HealthCheck method*), 17

E

evaluate_() (*vishack.HealthCheck method*), 17

G

generate_sample_config() (*in module vishack*), 15
get_alerts() (*vishack.HealthCheck method*), 17
get_name_and_ext() (*in module vishack.data.output*), 25
get_reference() (*vishack.data.diaggui.Diaggui method*), 23
get_results() (*vishack.data.diaggui.Diaggui method*), 24

H

HealthCheck (*class in vishack*), 15

I

items (*vishack.data.diaggui.Diaggui attribute*), 22

M

mae() (*in module vishack.core.evaluate*), 20
make_script() (*in module vishack.data.diag*), 22
measure() (*vishack.data.diaggui.Diaggui method*), 24
module
    vishack.core.evaluate, 20
    vishack.data.diag, 22
    vishack.data.diaggui, 22
    vishack.data.ezca, 18
    vishack.data.output, 25
mse() (*in module vishack.core.evaluate*), 20

P

parallel_read() (*in module vishack.data.ezca*), 19
parallel_time_average() (*in module vishack.data.ezca*), 19
parallel_time_series() (*in module vishack.data.ezca*), 19
path (*vishack.data.diaggui.Diaggui attribute*), 22
paths (*vishack.HealthCheck attribute*), 16
print_report() (*vishack.HealthCheck method*), 18
psd() (*vishack.data.diaggui.Diaggui method*), 24

R

reference_evaluate() (*vishack.HealthCheck method*), 18
rename() (*in module vishack.data.output*), 25
rename_method_123() (*in module vishack.data.output*), 25
rename_method_utc() (*in module vishack.data.output*), 26
report (*vishack.HealthCheck attribute*), 16
report_header (*vishack.HealthCheck attribute*), 16
report_to_string() (*vishack.HealthCheck method*), 18
rms() (*in module vishack.core.evaluate*), 20
run_measurement() (*in module vishack.data.diag*), 22

T

tf() (*vishack.data.diaggui.Diaggui method*), 24

V

vishack.core.evaluate
    module, 20

vishack.data.diag
    module, 22
vishack.data.diaggui
    module, 22
vishack.data.ezca
    module, 18
vishack.data.output
    module, 25

W

wmae() (*in module vishack.core.evaluate*), 21
wmse() (*in module vishack.core.evaluate*), 21
wrms() (*in module vishack.core.evaluate*), 21