

Running a response spectrum analysis

Table of contents

1 Before you start	1
2 Prepare the frequency step	1
3 Extract the modal data	2
4 Write a case file	2
5 Combine the modes	3
6 Read the results	3
7 Compare against Abaqus	4

A response spectrum analysis runs in three commands: extract the modal data from the output database, combine that data against a case file, and read the peak results. Only the first command needs Abaqus, so a single extraction supports as many spectra and summation rules as you want to study. The same extraction also drives a **random response analysis**, which takes a spectral density instead of a spectrum.

1 Before you start

The utility lives in the Coreform IGA for Abaqus source tree at `interop/abaqus/response_spectrum/`. It is not installed with the product, so add it to `PYTHONPATH` before running any command. The examples below use a shell variable for its location.

```
export RESPONSE_SPECTRUM_DIR=/path/to/cf/interop/abaqus/response_spectrum
export PYTHONPATH="$RESPONSE_SPECTRUM_DIR"
```

You also need NumPy for the combination step. The extraction step uses the NumPy that ships inside Abaqus Python, so it needs nothing extra.

2 Prepare the frequency step

Your output database must contain a `*FREQUENCY` step with mass-normalized eigenvectors, which is the default for the Lanczos and AMS eigensolvers. Participation factors are only consistent with that normalization, so a displacement-normalized extraction produces wrong peaks.

The frequency step must also write field output for every variable you want combined. The following step extracts 20 modes and records displacement, reaction force, and stress mode shapes:

```
*Step, name=FREQ, perturbation
*Frequency, eigensolver=Lanczos, normalization=mass
20,
*Output, field
*Node Output
U, RF
*Element Output, directions=YES
S
*End Step
```

You do not need to request **V** or **A**. Peak velocity and acceleration are derived from the displacement mode shapes using the velocity and acceleration spectra.

The eigenfrequencies, generalized masses, participation factors, and modal effective masses are written to the output database automatically as history data, so no extra output request is needed for them.

3 Extract the modal data

Run the `extract` command under `abaqus python`. This is the only step that reads the output database, and therefore the only step that needs an Abaqus license.

```
cd /path/to/your/job

abaqus python -m respspec extract myjob.odb myjob_modal.npz --fields U,S,RF
```

The command writes `myjob_modal.npz`, a single NumPy archive holding the eigenfrequencies, participation factors, generalized masses, and the requested mode-shape fields.

Pass only the fields the frequency step actually wrote, because a missing field is an error. Add `--frequency-step NAME` when the job contains more than one frequency extraction; the first one is used by default.

Confirm the extraction before going further:

```
python3 -m respspec info myjob_modal.npz
```

The report lists each mode with its frequency, translational participation factors, and generalized mass, followed by the available fields. Generalized masses equal to 1 confirm the modes are mass normalized.

4 Write a case file

A case file is a JSON document that defines your spectra, the excitation directions, the summation rules, and the modal damping. Start from the example below, or from one

of the eight validation cases defined in `abaqus_models/validation_cases.py`, which the validation pipeline writes out as JSON under `abaqus_models/jobs/cases/`.

```
{
  "spectra": {
    "DESIGN": {
      "type": "ACCELERATION",
      "gravity": null,
      "rows": [[0.8, 0.5, 0.05], [22.0, 50.0, 0.05], [5.0, 5000.0, 0.05]]
    }
  },
  "excitations": [
    {"spectrum": "DESIGN", "direction": [1, 0, 0], "scale": 1.0, "duration": null},
    {"spectrum": "DESIGN", "direction": [0, 1, 0], "scale": 0.7, "duration": null}
  ],
  "modal_summation": "CQC",
  "directional_summation": "SRSS",
  "damping": 0.05
}
```

Each spectrum row is a magnitude, a frequency in cycles per time, and a damping ratio, in the same order as `*SPECTRUM` data lines. Excitation directions must be unit vectors, and they must be mutually orthogonal. See the [case file reference](#) for every field and the values it accepts.

5 Combine the modes

The `combine` command applies the case file to the extracted modal data. It runs under an ordinary Python interpreter, with no Abaqus process and no license.

```
python3 -m respspec combine myjob_modal.npz mycase.json myjob_peaks.npz --fields U,V,A,S
```

The command prints the number of output points and the largest magnitude for each field, then writes the peaks to `myjob_peaks.npz`. Run it again with a different case file whenever you want another spectrum or summation rule; the extraction does not have to be repeated.

6 Read the results

The peaks are stored as a NumPy archive rather than an output database, so read them in Python. Each field carries the labels that identify its output points, which lets you map a value back to a node or an element.

```
import numpy as np
from respspec.modal_data import load_peak_fields_npz

peaks = load_peak_fields_npz("myjob_peaks.npz")

displacement = peaks["U"]
values = displacement.values[0]          # (n_points, n_components)
worst = int(np.argmax(np.linalg.norm(values, axis=1)))

print(displacement.component_labels)      # ('U1', 'U2', 'U3')
print(int(displacement.node_labels[worst]), values[worst])
```

Stress is returned as components at integration points, labelled by element and integration point.

! Important

Combine stress components, never invariants. Every summation rule discards sign, so a peak von Mises stress computed from already-combined components is the correct result, while combining per-mode von Mises values is not.

7 Compare against Abaqus

When the same job also contains a solver-computed `*RESPONSE SPECTRUM` step, you can extract those results and check the Python combination against them. Add `--peaks` to the extraction to write one archive per response spectrum step:

```
abaqus python -m respspec extract myjob.odb myjob_modal.npz \
    --fields U,S,RF --peaks myjob_peaks_abaqus --peak-fields U,V,A,S
```

Then compare the two sets of peaks field by field:

```
python3 -m respspec compare myjob_peaks.npz myjob_peaks_abaqus/
STEPNAME_peaks.npz
```

The report gives, for each field, the largest absolute difference, that difference scaled by the peak value of the field, and the worst point-wise relative error:

```
S: points= 9400 ref_max= 3.58890e+06 max_abs= 4.501e-01 scaled= 1.25e-07
pointwise_rel= 1.85e-07 worst@896/S11
PASS: worst scaled error 1.254e-07 (tolerance 1.0e-04)
```

The command exits with a nonzero status when any field exceeds the tolerance, so it can be used directly in a script. Use `--tolerance` to change the threshold.

Note that `RF` is absent from the comparison. Abaqus does not write reaction forces in a SIM-based response spectrum step, so there is nothing to compare them against.