Created in COMSOL Multiphysics 6.2

Eigenmodes of a Viscoelastic Structural Damper

Introduction

The example studies the natural frequencies and corresponding eigenmodes of a typical viscoelastic damper. Damping elements involving layers of viscoelastic materials are often used for the reduction of seismic and wind induced vibrations in buildings and other tall structures. The common feature for such structures is that the frequency of their possible forced vibrations is low. Thus, it is important to use dampers with natural frequencies that are high enough to avoid any failures due to resonances.

Solving for eigenfrequencies in a structure where the deformation to a large extent is controlled by a viscoelastic material (or any other material which has frequency dependent material properties) requires special techniques. A standard eigenfrequency problem without damping can be formulated as

$$[\mathbf{K} - f^2\mathbf{M}]\mathbf{u} = 0 \tag{1}$$

where **K** is the stiffness matrix, **M** is the mass matrix, **u** is the eigenmode displacement vector, and *f* is the frequency. In most cases, **K** is independent of frequency, but for a viscoelastic material the eigenvalue equation actually is

$$[\mathbf{K}(f) - f^2\mathbf{M}]\mathbf{u} = 0 \tag{2}$$

In this example, it is shown how you can handle this type of problem.

Model Definition

The geometry of the viscoelastic damper is shown in Figure 1 (from Ref. 1). The damper consists of two layers of viscoelastic material confined between mounting elements made of steel.

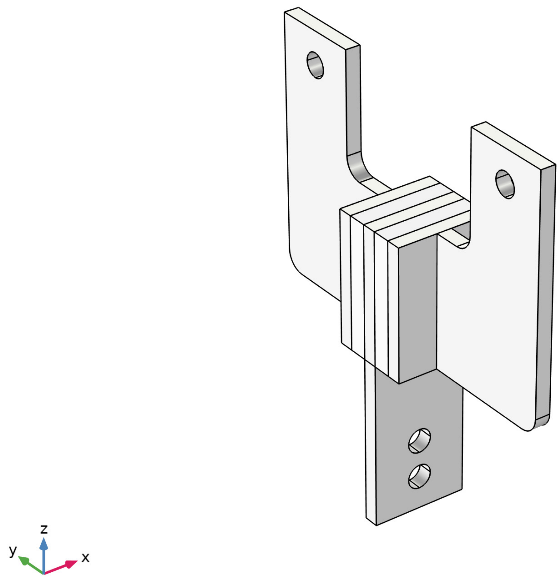


Figure 1: Viscoelastic damping element.

One of the mounting elements is modeled as fixed, and two other elements are partially constrained to represent a typical operating regime of the damper.

The viscoelastic layers are modeled using complex valued material data. For frequency domain and eigenfrequency analyses, the frequency decomposition for the deviatoric stress and strain tensors is in general performed as:

$$S_{\mathrm{d}} = \hat{S}_{\mathrm{d}} e^{2\pi i f t}$$

$$\varepsilon_{\mathrm{d}} = \hat{\varepsilon}_{\mathrm{d}} e^{2\pi i f t}$$

where f is the frequency.

The deviatoric stress is related then to the strain as

$$\hat{S}_{\mathrm{d}} = 2(G'(f) + iG''(f))\hat{\varepsilon}_{\mathrm{d}} \tag{3}$$

where the G' and G'' are the storage and loss moduli, respectively.

In this example, the moduli are specified by their reference values given at two reference frequencies f_{r1} = 200 Hz and f_{r2} =1000 Hz:

TABLE 1: STORAGE AND LOSS MODULI.

f	f_{r1}	f_{r2}
G'	3.0848E7 Pa	7.8348E7 Pa
G''	3.6551E7 Pa	8.4935E7 Pa

The frequency dependency is approximated by straight lines in the log-log space using the above data. Thus, the following expressions are used for the moduli:

$$G'(f) = G'(f_{r1})\left(\frac{f}{f_{r1}}\right)^n$$

where

$$n = \log\left(\frac{G'(f_{r2})}{G'(f_{r1})}\right) \Big/ \log\left(\frac{f_{r2}}{f_{r1}}\right)$$

and

$$G''(f) = G''(f_{r1})\left(\frac{f}{f_{r1}}\right)^m$$

where

$$m = \log\left(\frac{G''(f_{r2})}{G''(f_{r1})}\right) \Big/ \log\left(\frac{f_{r2}}{f_{r1}}\right)$$

Substituting the deviatoric stress given by Equation 3 into the equation of motion gives

$$-\rho 4\pi^2 f^2 \hat{\mathbf{u}} = -\nabla\hat{p} + 2(G'(f) + iG''(f))\nabla \cdot \hat{\varepsilon}_{\mathrm{d}}$$

which, together with the boundary conditions, will result in a nonlinear eigenvalue problem for f. The eigenvalue problem will determine the natural frequencies of the system.

COMSOL Multiphysics solves such nonlinear problems by expanding all expressions containing the frequency down to quadratic polynomials using a frequency linearization point f_L which you can specify in the **Eigenvalue Solver** node (100 Hz is used by default).

The eigenvalue problem, which is solved, is then

$$[\mathbf{K}(f_L) - f^2\mathbf{M}]\mathbf{u} = 0$$

Thus, the results become dependent on the choice of the frequency linearization point. Selecting the point closer to one of the actual natural frequencies will produce a better result for that particular frequency. Hence, to obtain more accurate eigenfrequencies, they need to be computed one by one, using a certain iterative process.

Results and Discussion

In this model, six eigenfrequencies are initially computed using the default value of 100 Hz as the frequency linearization point. This allows you to identify the frequency range 220 – 1000 Hz for further investigations. The computed eigenfrequencies can only be expected to be correct by the order of magnitude.

As an initial refinement step, eight frequency linearization points, f_i, are evenly spaced along the real axis:

$$f_1 = 250\ \text{Hz}, f_{i+1} = f_i + 100\ \text{Hz}$$

The solver is then set to search for eigenvalues with real part in the interval: [f_i – 50 Hz, f_i + 50 Hz], and imaginary part in the interval: [0, 1000 Hz]. Six eigenfrequencies are found after a sweep over all eight intervals.

In the second refinement step, the six eigenfrequencies computed during the initial refinement step are used as new frequency linearization points, f_i. Now the solver is set to search for eigenvalues with the imaginary part in the interval $[0, 2\ \mathrm{Im}(f_i)]$, and the real part in the interval $[0.5\ \mathrm{Re}(f_{i-1} + f_i), 0.5\ \mathrm{Re}(f_i + f_{i+1})]$ to completely cover the intervals between the linearization points.

[Figure 2](#) shows the eigenfrequencies computed at different steps of the update algorithm.

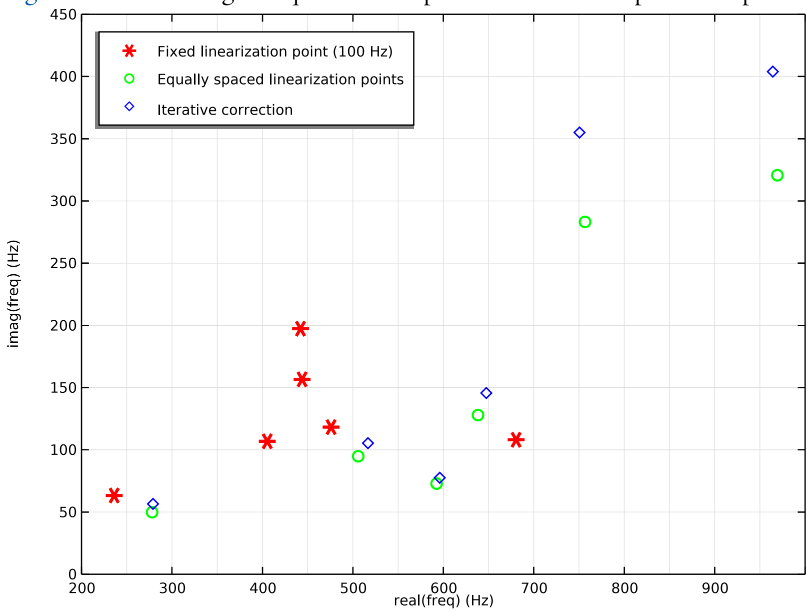


Figure 2: The eigenfrequency distribution in the complex plane. The red markers show the values computed using a fixed frequency linearization point. The green markers show the values computed using linearization points evenly spaced along the real axis. The blue markers show the values updated using the previously computed frequencies as linearization points.

Thus, a uniform distribution of the linearization points is capable of producing a decent estimate for the real (periodic) parts of the eigenfrequencies, but it fails to correctly predict the values of their imaginary (damping) parts. This justifies the need for the refinement step. Further updates of the same type did not show any substantial change in the eigenfrequency values.

The first eigenmodes computed at the first and last steps of the algorithm are shown in Figure 3 and Figure 4, respectively.

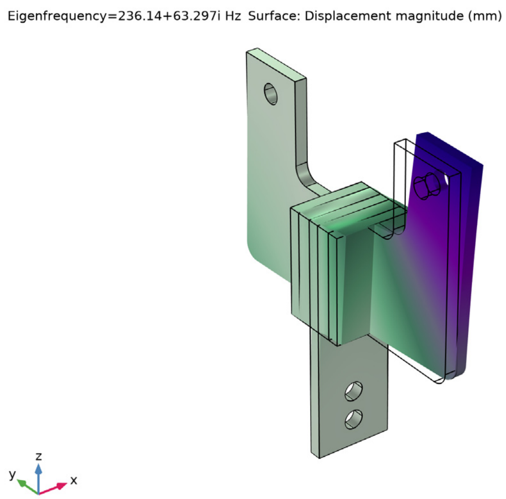


Figure 3: First eigenmode computed using the default frequency linearization point.

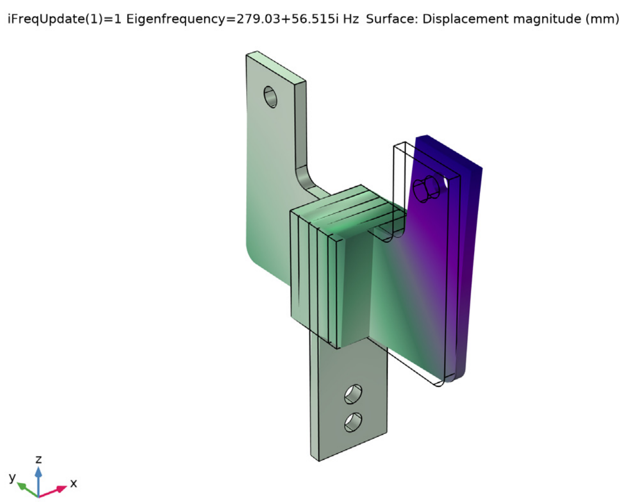


Figure 4: First eigenmode computed using the updated values for the frequency linearization point.

Notes About the COMSOL Implementation

You model in 3D using the Solid Mechanics interface with a **Linear Elastic Material** and add the **Viscoelasticity** node to the domains representing the viscoelastic layers.

To set up and perform an iterative eigenfrequency update, you use two model methods. A parametric sweep including all study steps is auto-generated using one of the methods. The other method performs all necessary evaluations and solver parameter updates between the parametric steps. You enter the Java® code for the methods using the method editor which is only available in the Windows® version of the COMSOL Desktop®. More comments about the methods can be found in the modeling step-by-step instructions below.

References

1. S.W. Park “Analytical Modeling of Viscoelastic Dampers for Structural and Vibration Control,” *Int. J. Solids and Structures*, vol. 38, pp. 694–701, 2001.

2. K.L. Shen and T.T. Soong, “Modeling of Viscoelastic Dampers for Structural Applications,” *J. Eng. Mech.*, vol. 121, pp. 694–701, 1995.

Application Library path: `Structural_Mechanics_Module/Dynamics_and_Vibration/viscoelastic_damper_eigenmodes`

Modeling Instructions

From the **File** menu, choose **New**.

NEW

In the **New** window, click **Model Wizard**.

MODEL WIZARD

1 In the **Model Wizard** window, click **3D**.

2 In the **Select Physics** tree, select **Structural Mechanics>Solid Mechanics (solid)**.

3 Click **Add**.

4 Click **Study**.

5 In the **Select Study** tree, select **General Studies>Eigenfrequency**.

6 Click Done.

GLOBAL DEFINITIONS

Parameters 1

Import the viscoelastic material data from a file.

1 In the **Model Builder** window, under **Global Definitions** click **Parameters 1**.

2 In the **Settings** window for **Parameters**, locate the **Parameters** section.

3 Click **Load from File**.

4 Browse to the model's Application Libraries folder and double-click the file `viscoelastic_damper_eigenmodes_param.txt`.

The data contains the reference values of the storage and loss moduli given at two reference frequencies. You approximate the data by straight lines in the log-log space. This can be done by using analytic functions as follows.

Analytic 1 (an1)

1 In the **Home** toolbar, click f(x) **Functions** and choose **Global>Analytic**.

2 In the **Settings** window for **Analytic**, type `gstor` in the **Function name** text field.

3 Locate the **Definition** section. In the **Expression** text field, type `gsr1*(f/fr1)^ns`.

4 In the **Arguments** text field, type `f`.

5 Locate the **Units** section. In the **Function** text field, type `Pa`.

6 In the table, enter the following settings:

Argument	Unit
f	Hz

7 Click to expand the **Advanced** section. Select the **May produce complex output for real arguments** check box.

8 Locate the **Plot Parameters** section. In the table, enter the following settings:

Plot	Argument	Lower limit	Upper limit	Fixed value	Unit
√	f	fr1	fr2	0	Hz

Analytic 1 (gstor)

Right-click **Analytic 1 (gstor)** and choose **Duplicate**.

Analytic 2 (gstor2)

1 In the **Model Builder** window, under **Global Definitions** click **Analytic 2 (gstor2)**.

2 In the **Settings** window for **Analytic**, type `gloss` in the **Function name** text field.

3 Locate the **Definition** section. In the **Expression** text field, type `glr1*(f/fr1)^nl`.

GEOMETRY 1

Import the predefined geometry from a file.

1 In the **Geometry** toolbar, click **Insert Sequence** and choose **Insert Sequence**.

2 Browse to the model's Application Libraries folder and double-click the file `viscoelastic_damper_geom_sequence.mph`.

The imported geometry should look similar to that shown in Figure 1.

SOLID MECHANICS (SOLID)

Linear Elastic Material 2

1 In the **Model Builder** window, under **Component 1 (comp1)** right-click **Solid Mechanics (solid)** and choose **Material Models>Linear Elastic Material**.

2 In the **Settings** window for **Linear Elastic Material**, locate the **Linear Elastic Material** section.

3 From the **Specify** list, choose **Bulk modulus and shear modulus**.

4 From the **Use mixed formulation** list, choose **Pressure formulation**.

5 Select Domains 2 and 5 only.

Viscoelasticity 1

1 In the **Physics** toolbar, click **Attributes** and choose **Viscoelasticity**.

Use the analytic functions to enter the viscoelastic moduli as functions of frequency.

2 In the **Settings** window for **Viscoelasticity**, locate the **Viscoelasticity Model** section.

3 From the **Material model** list, choose **User defined**.

4 In the G' text field, type `gstor(solid.freq)`.

5 In the G'' text field, type `gloss(solid.freq)`.

ADD MATERIAL

1 In the **Home** toolbar, click **Add Material** to open the **Add Material** window.

2 Go to the **Add Material** window.

3 In the tree, select **Built-in>Steel AISI 4340**.

4 Click **Add to Component** in the window toolbar.

MATERIALS

Viscoelastic

1 In the **Model Builder** window, under **Component 1 (comp1)** right-click **Materials** and choose **Blank Material**.

2 In the **Settings** window for **Material**, type `Viscoelastic` in the **Label** text field.

3 Select Domains 2 and 5 only.

4 Locate the **Material Contents** section. In the table, enter the following settings:

Property	Variable	Value	Unit	Property group
Bulk modulus	K	4e8	N/m²	Bulk modulus and shear modulus
Shear modulus	G	5.86e4	N/m²	Bulk modulus and shear modulus
Density	rho	1060	kg/m³	Basic

5 In the **Home** toolbar, click **Add Material** to close the **Add Material** window.

6 In the **Settings** window for **Materials**, in the **Graphics** window toolbar, click ▼ next to **Colors**, then choose **Show Material Color and Texture**.

SOLID MECHANICS (SOLID)

Fixed Constraint 1

1 In the **Physics** toolbar, click **Boundaries** and choose **Fixed Constraint**.

2 In the **Settings** window for **Fixed Constraint**, locate the **Boundary Selection** section.

3 From the **Selection** list, choose **Bottom Holes**.

Prescribed Displacement 1

1 In the **Physics** toolbar, click **Boundaries** and choose **Prescribed Displacement**.

2 In the **Settings** window for **Prescribed Displacement**, locate the **Boundary Selection** section.

3 From the **Selection** list, choose **Right Hole**.

4 Locate the **Prescribed Displacement** section. From the **Displacement in x direction** list, choose **Prescribed**.

5 From the **Displacement in y direction** list, choose **Prescribed**.

Prescribed Displacement 2

1 In the **Physics** toolbar, click **Boundaries** and choose **Prescribed Displacement**.

2 In the **Settings** window for **Prescribed Displacement**, locate the **Boundary Selection** section.

3 From the **Selection** list, choose **Left Hole**.

4 Locate the **Prescribed Displacement** section. From the **Displacement in y direction** list, choose **Prescribed**.

MESH 1

Free Quad 1

1 In the **Mesh** toolbar, click **More Generators** and choose **Free Quad**.

2 Select Boundary 30 only.

Size 1

1 Right-click **Free Quad 1** and choose **Size**.

2 In the **Settings** window for **Size**, locate the **Element Size** section.

3 From the **Predefined** list, choose **Finer**.

Distribution 1

1 In the **Model Builder** window, right-click **Free Quad 1** and choose **Distribution**.

2 Select Edge 65 only.

Swept 1

1 In the **Mesh** toolbar, click **Swept**.

2 In the **Settings** window for **Swept**, locate the **Domain Selection** section.

3 From the **Geometric entity level** list, choose **Domain**.

4 Select Domain 7 only.

Distribution 1

1 Right-click **Swept 1** and choose **Distribution**.

2 In the **Settings** window for **Distribution**, locate the **Distribution** section.

3 In the **Number of elements** text field, type 2.

Free Quad 2

1 In the **Mesh** toolbar, click **More Generators** and choose **Free Quad**.

2 Select Boundaries 2 and 61 only.

Size 1

1 Right-click **Free Quad 2** and choose **Size**.

2 In the **Settings** window for **Size**, locate the **Element Size** section.

3 From the **Predefined** list, choose **Fine**.

Swept 2

1 In the **Mesh** toolbar, click **Swept**.

2 In the **Settings** window for **Swept**, locate the **Domain Selection** section.

3 From the **Geometric entity level** list, choose **Domain**.

4 Select Domains 1, 2, and 4 only.

Distribution 1

1 Right-click **Swept 2** and choose **Distribution**.

2 In the **Settings** window for **Distribution**, locate the **Distribution** section.

3 In the **Number of elements** text field, type 2.

Copy Domain 1

1 In the **Model Builder** window, right-click **Mesh 1** and choose **Copying Operations>Copy Domain**.

2 Select Domains 1, 2, and 7 only.

3 In the **Settings** window for **Copy Domain**, locate the **Destination Domains** section.

4 Click to select the **Activate Selection** toggle button.

5 Select Domains 5, 6, and 8 only.

Free Quad 3

1 In the **Mesh** toolbar, click **More Generators** and choose **Free Quad**.

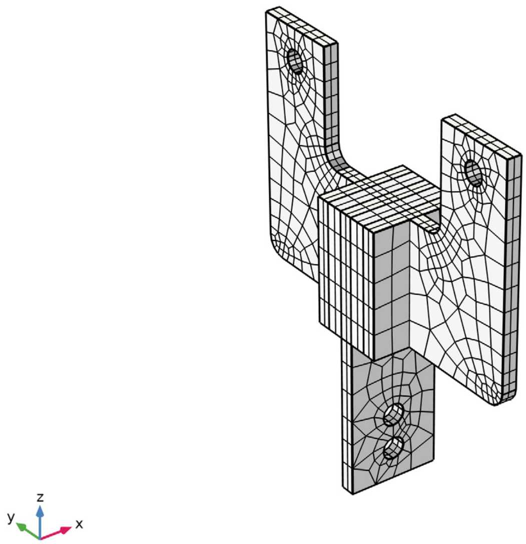
2 Select Boundary 10 only.

Swept 3

1 In the **Mesh** toolbar, click **Swept**.

2 In the **Settings** window for **Swept**, click **Build All**.

The complete mesh should look similar to that shown in the figure below.



Perform the initial eigenfrequency analysis.

STUDY 1

1 In the **Home** toolbar, click **Compute**.

Evaluate the computed eigenvalues into a table.

These values are not exact because of their distance from the frequency linearization point (by default, set to 100 Hz). However, they can indicate a region of the real and imaginary parts for the eigenfrequency update analysis.

RESULTS

Eigenfrequencies (Study 1)

1 In the **Model Builder** window, under **Results** click **Eigenfrequencies (Study 1)**.

2 In the **Eigenfrequencies (Study 1)** toolbar, click **Evaluate**.

In the remaining part of the modeling, you will create model methods. Note that the method editor is only available in the Windows® version of the COMSOL Desktop.

ROOT

1 From the **File** menu, choose **Preferences**.

2 In the **Preferences** dialog box, select **Application Builder>Methods** in the tree.

3 Clear the **View all code** check box.

4 Click **OK**.

APPLICATION BUILDER

In the **Home** toolbar, click A **Application Builder**.

METHODS

utilUpdate

1 In the **Home** toolbar, click **More Libraries** and choose **Utility Class**.

2 Right-click **util1** and choose **Rename**.

3 In the **Rename Utility Class** dialog box, type `utilUpdate` in the **New name** text field.

4 Click **OK**.

5 Right-click **utilUpdate** and choose **Edit**.

6 Copy the following code into the **utilUpdate** window:

```
// Use a dedicated parameter group for method-generated parameters
 public static String parGroupUpdateTag = "parGroupUpdate";

 // use a dedicated table to store and update the computed eigenfrequencies
 public static String tableUpdateTag = "tableUpdate";

 /** This method will update the parameters before each parametric step.
 The parameters are used in the eigenvalue solver configuration to specify the
linearization point and search method.
 The method can be used during both the initial and update parametric sweeps. */
 public static void solverParametersUpdate() {
 ModelParamGroup parGroup = model.param(parGroupUpdateTag);
 int i = Double.valueOf(parGroup.get("iFreqUpdate")).intValue();
 int nFreq = Double.valueOf(parGroup.get("numFreqUpdate")).intValue();
 if (parGroup.get("statusUpdate").equals("0")) {
 /* Initial sweep.
 The frequency linearization points are evenly spaced along the real axis:
realFreq(i+1) = realFreq(i) + 2*Lplus
 using the same value of Lplus and all imaginary parts being zero.
 The solver will search for eigenvalues with real parts in the interval
[realFreq(i)-Lplus, Freq(i)+Lplus]
 and imaginary parts in the interval [0, 2*imagFreqMax]. */
 double frMin = Double.valueOf(parGroup.get("frMinUpdate"));
 double Lp = Double.valueOf(parGroup.get("LplusUpdate"));
 parGroup.set("realFreqUpdate", String.valueOf(frMin+Lp*(2*i-1)));
 } else {
 /* Update sweep.
```

```
 The frequency linearization points are taken from a table precomputed during
the previous sweep.
 The solver will search for eigenvalues with real parts in the interval
[realFreq(i)-Lminus, Freq(i)+Lplus],
 where Lminus and Lplus may vary from step to step, and imaginary parts in the
interval [0, 2*imagFreq(i)]. */
 TableFeature tableUpdate = model.result().table(tableUpdateTag);
 String[][] tableData = tableUpdate.getTableData(false);
 String fr = tableData[i-1][2];
 String fi = tableData[i-1][3];
 parGroup.set("realFreqUpdate", fr);
 parGroup.set("imagFreqUpdate", fi);
 parGroup.set("fiMaxUpdate", "2*"+fi);
 String Lm;
 String Lp;
 if (i > 1) {
 // Lplus from the previous interval becomes new Lminus
 Lm = parGroup.get("LplusUpdate");
 parGroup.set("LminusUpdate", Lm);
 }
 if (i < nFreq) {
 // take the next computed value
 String fr1 = tableData[i][2];
 // use half a distance to it as Lplus
 Lp = String.valueOf((Double.valueOf(fr1)-Double.valueOf(fr))/2);
 if (i == 1) // use the same values for Lminus for the first frequency
 Lm = Lp;
 else // Lplus from the previous interval becomes new Lminus
 Lm = parGroup.get("LplusUpdate");
 } else { // i == nFreq
 // Lplus from the previous interval becomes new Lminus
 Lm = parGroup.get("LplusUpdate");
 // Use the same value for new Lplus for the last frequency
 Lp = Lm;
 }
 parGroup.set("LplusUpdate", Lp);
 parGroup.set("LminusUpdate", Lm);
 }
 }

 /* Run a parametric sweep. Control possible error due to empty solution. */
 public static void runSweep(BatchFeature batch) {
 // do not stop on errors, it will continue to the next step in sweep
 batch.set("err", false);
 try {
 batch.run();
 }
 catch (com.comsol.util.exceptions.FlException ex) {
 com.comsol.util.classes.FlStringList messages = ex.getMessages();
 // ignore the errors if no eigenvalues was found for some step the sweep
 if (!(messages.contains("Error_in_sweep") &&
messages.contains("No_eigenfrequencies_found")))
 throw ex; // some other cause, pass the error through
 }
```

```
}
```

GLOBAL METHOD

1 In the **Home** toolbar, click **New Method** and choose **Global Method**.

2 In the **Global Method** dialog box, type `solverParametersUpdate` in the **Name** text field.

3 Click **OK**.

4 Copy the following code into the **solverParametersUpdate** window:

```
utilUpdate.solverParametersUpdate();
```

5 In the **Home** toolbar, click **New Method** and choose **Global Method**.

6 In the **Global Method** dialog box, type `runStudyUpdate` in the **Name** text field.

7 Click **OK**.

runStudyUpdate

1 In the **Application Builder** window, under **Methods** click **runStudyUpdate**.

2 In the **Settings** window for **Method**, locate the **Inputs and Output** section.

3 Find the **Inputs** subsection. Click + **Add**.

4 In the table, enter the following settings:

Name	Type	Default	Description	Unit
realFreqMin	Double	220	Minimum frequency	Hz

5 Click + **Add**.

6 In the table, enter the following settings:

Name	Type	Default	Description	Unit
realFreqMax	Double	1000	Maximum frequency	Hz

7 Click + **Add**.

8 In the table, enter the following settings:

Name	Type	Default	Description	Unit
imagFreqMax	Double	500	Maximum frequency imaginary part	Hz

9 Click + **Add**.

10 In the table, enter the following settings:

Name	Type	Default	Description	Unit
numFreq	Integer	5	Approximate number of eigenfrequencies	

11 Copy the following code into the **runStudyUpdate** window:

```
/** This method will set up a special parametric study to perform the
eigenfrequency computations
 * for nonlinear eigenvalue problems.
 *
 * These four parameters are method inputs: realFreqMin, realFreqMax, numFreq,
imagFreqMax
 *
 * The method will search for eigenfrequencies with real part in interval:
[realFreqMin, realFreqMax]
 * and imaginary parts in interval: [0, imagFreqMax].
 * The initial approximate number of eigenfrequencies is taken as: numFreq.
 *
 * The method will perform two parametric sweeps with respect to the frequency
number: i.
 *
 * During the initial sweep, the linearization point frequencies are even-spaced
along the real axis:
 * imagFreq(i) = 0
 * realFreq(1) = realFreqMin + Lplus
 * realFreq(numFreq) = realFreqMax - Lplus
 * realFreq(i+1) = realFreq(i) + 2*Lplus
 * Lplus = 0.5*(realFreqMax-realFreqMin)/(nFreq-1)
 *
 * One each parametric step, the solver will search for eigenvalues with real
parts in the interval:
 * [realFreq(i)-Lpus, Freq(i)+Lplus]
 * and imaginary part in the interval: [0, imagFreqMax].
 *
 * During the update sweep, the linearization point frequencies are taken from
a table precomputed during the previous sweep.
 * One each parametric step, the solver will search for eigenvalues with real
parts in the interval:
 * [realFreq(i)-Lminus, realFreq(i)+Lplus]
 * Lminus = (realFreq(i)-realFreq(i-1))/2
 * Lplus = (realFreq(i+1)-realFreq(i))/2
 * and imaginary part in the interval: [0, 2*imagFreq(i)].
 * */

 String frMin = String.valueOf(realFreqMin);
 String frMax = String.valueOf(realFreqMax);
 int nFreq = numFreq;
 String fiMax = String.valueOf(imagFreqMax);

 // use a dedicated group for method-generated parameters
 String parGroupTag = utilUpdate.parGroupUpdateTag;
 if (model.param().group().index(parGroupTag) ModelParamGroup )
```

```
model.param().group().remove(parGroupTag);
ModelParamGroup parGroup = model.param().group().create(parGroupTag);
parGroup.label("Parameters Eigenfrequency Update (autogenerated)");
parGroup.comments("This parameter group has been autogenerated.");

// status parameter: Update==0 for the initial sweep, Update==1 for the update
sweep
parGroup.set("statusUpdate", "0");

// parameters to be used in the solver settings and parametric sweeps
parGroup.set("numFreqUpdate", String.valueOf(nFreq));
parGroup.set("iFreqUpdate", "1");
parGroup.set("realFreqUpdate", frMin);
parGroup.set("imagFreqUpdate", "0");
parGroup.set("FreqUpdate", "realFreqUpdate+i*imagFreqUpdate");
parGroup.descr("FreqUpdate", "Frequency parameter");
parGroup.descr("realFreqUpdate", "Real part");
parGroup.descr("imagFreqUpdate", "Imaginary part");
parGroup.set("frMinUpdate", frMin);
parGroup.set("fiMaxUpdate", fiMax);

double Lp = 0.5*(Double.valueOf(frMax)-Double.valueOf(frMin))/nFreq;
parGroup.set("LminusUpdate", String.valueOf(Lp));
parGroup.set("LplusUpdate", String.valueOf(Lp));

// set up a dedicated study for the frequency sweep
String eigStudyTag = "studyEigenfrequencyUpdate";
Study eigStudy;
String eigStudyStepTag = "eigStudyStep";
StudyFeature eigStudyStep;
if (model.study().index(eigStudyTag) == -1) { // no such study
eigStudy = model.study().create(eigStudyTag);
eigStudy.label("Study Eigenfrequency Update (autogenerated)");
eigStudy.comments("This study has been autogenerated.");
eigStudyStep = eigStudy.create(eigStudyStepTag, "Eigenfrequency");
} else {
eigStudy = model.study(eigStudyTag);
if (eigStudy.feature().index(eigStudyStepTag) == -1) // no such study step
eigStudyStep = eigStudy.create(eigStudyStepTag, "Eigenfrequency");
else
eigStudyStep = eigStudy.feature(eigStudyStepTag);
}

// configure the eigenfrequency study step to search for eigenvalues in a region
eigStudyStep.set("eigmethod", "region");
eigStudyStep.set("appnreigs", 1);
eigStudyStep.set("maxnreigs", 3);
eigStudyStep.set("chkeigregion", false);
eigStudyStep.set("eigsr", "realFreqUpdate-LminusUpdate");
eigStudyStep.set("eiglr", "realFreqUpdate+LplusUpdate");
eigStudyStep.set("eigli", "fiMaxUpdate");

// set up a parametric sweep with respect to the frequency number
String paramSweepTag = "paramSweep";
StudyFeature paramSweep;
```

```
if (eigStudy.feature().index(paramSweepTag) == -1) // no such feature
paramSweep = eigStudy.create(paramSweepTag, "Parametric");
else
paramSweep = eigStudy.feature(paramSweepTag);
paramSweep.setIndex("pname", "iFreqUpdate", 0);
paramSweep.setIndex("plistarr", "range(1,1,"+nFreq+")", 0);
paramSweep.set("paramselect", false);

// generate the solvers if needed
String[] eigSolTags = eigStudy.getSolverSequences("All");
if (eigSolTags.length == 0) {
eigStudy.showAutoSequences("sol");
eigStudy.showAutoSequences("jobs");
eigSolTags = eigStudy.getSolverSequences("All");
}

// set the linearization point in the eigenvalue solver
SolverSequence eigenvalueSolver = model.sol(eigSolTags[0]);
eigenvalueSolver.feature("e1").set("transeigref", true);
eigenvalueSolver.feature("e1").set("eigref", "FreqUpdate");
eigenvalueSolver.feature("e1").feature("dDef").set("linsolver", "pardiso");

// find the auto-generated batch that corresponds to the parametric sweep
String batchTag = "p1"; // most probable candidate
for (PropFeature batch : model.batch()) {
if (batch.getString("control").equals(paramSweepTag)) {
batchTag = batch.tag();
break;
}
}
BatchFeature batch = model.batch(batchTag);

// add a model method call to the batch to set up the parameter before each
parametric step
String methodCallTag = "methodCall";
BatchFeature methodCall;
if (batch.feature().index(methodCallTag) == -1) // no such feature
methodCall = batch.feature().create(methodCallTag, "Methodcall");
else
methodCall = batch.feature(methodCallTag);
methodCall.set("ref", "methodcallSolverParametersUpdate");
// move the method up to place it before the parametric step
batch.feature().move("methodCall", 0);

// perform the initial sweep computation
utilUpdate.runSweep(batch);

// find the auto-generated dataset that corresponds to the pametric sweep
Results results = model.result();
String datasetTag = "dset3"; // most probable candidate
for (DatasetFeature dataset : results.dataset()) {
if
(dataset.getString("solution").equals(eigStudy.getSolverSequences("Parametric"
)[0])) {
datasetTag = dataset.tag();
```

```
break;
}
}

// evaluate the eigenfrequency into a table
String gevFreqTag = "gevFreqUpdate";
NumericalFeature gevFreq;
if (results.numerical().index(gevFreqTag) == -1) // no such feature
gevFreq = results.numerical().create(gevFreqTag, "EvalGlobal");
else
gevFreq = results.numerical(gevFreqTag);
gevFreq.set("data", datasetTag);
gevFreq.setIndex("expr", "real(freq)", 0);
gevFreq.setIndex("expr", "imag(freq)", 1);

TableFeatureList tableList = model.result().table();
String tableInitTag = "tableInit";
TableFeature tableInit;
if (tableList.index(tableInitTag) == -1) { // no such table
tableInit = tableList.create(tableInitTag, "Table");
tableInit.label("Table "+tableInitTag+" (init sweep)");
} else
tableInit = model.result().table(tableInitTag);
tableInit.clearTableData();

gevFreq.set("table", tableInitTag);
gevFreq.setResult();

// prepare the update sweep
parGroup.set("statusUpdate", "1");

// duplicate the frequency table
String tableUpdateTag = utilUpdate.tableUpdateTag;
if (tableList.index(tableUpdateTag) > -1) // table already exists
tableList.remove(tableUpdateTag);
TableFeature tableUpdate = tableList.duplicate(tableUpdateTag, tableInitTag);
tableUpdate.label("Table "+tableUpdateTag+" (update sweep)");
gevFreq.set("table", tableUpdateTag);
nFreq = tableUpdate.getNRows();
parGroup.set("numFreqUpdate", nFreq);

// perform the update sweep computations
eigStudyStep.set("maxnreigs", 1);
utilUpdate.runSweep(batch);

// update the eigenfrequency table
tableUpdate.clearTableData();
gevFreq.setResult();
int nFreqUpdate = tableUpdate.getNRows();
parGroup.set("numFreqUpdate", nFreqUpdate);
paramSweep.setIndex("plistarr", "range(1,1,"+nFreqUpdate+")", 0);

// repeat the update sweep until the same number of eigenfrequencies is found
in the consequent update
while (nFreq != nFreqUpdate) {
```

```
utilUpdate.runSweep(batch);
nFreq = nFreqUpdate;
tableUpdate.clearTableData();
gevFreq.setResult();
nFreqUpdate = tableUpdate.getNRows();
parGroup.set("numFreqUpdate", nFreqUpdate);
paramSweep.setIndex("plistarr", "range(1,1,"+nFreqUpdate+")", 0);
}

// finally add the updated eigenfrequencies as new parameters in the model
String[][] tableData = tableUpdate.getTableData(false);
for (int i = 0; i < nFreqUpdate; i++) {
parGroup.set("FreqUpdate"+String.valueOf(i+1), tableData[i][2]+"+i*"+
tableData[i][3]);
}
```

METHODS

1 In the **Home** toolbar, click **Model Builder** to switch to the main desktop.

2 Click **Method Call** and choose **solverParametersUpdate**.

GLOBAL DEFINITIONS

SolverParametersUpdate 1

1 In the **Model Builder** window, under **Global Definitions** click **SolverParametersUpdate 1**.

2 In the **Settings** window for **Method Call**, type `methodcallSolverParametersUpdate` in the **Tag** text field.

3 In the **Home** toolbar, click **Method Call** and choose **runStudyUpdate**.

RunStudyUpdate 1

1 In the **Model Builder** window, click **RunStudyUpdate 1**.

2 In the **Settings** window for **Method Call**, type `methodcallRunStudyUpdate` in the **Tag** text field.

In the **Home** toolbar, click **Run Method Call** and choose **RunStudyUpdate 1**.

RESULTS

Global Evaluation 1

1 In the **Model Builder** window, expand the **Eigenfrequencies (Study 1)** node, then click **Global Evaluation 1**.

2 In the **Settings** window for **Global Evaluation**, locate the **Expressions** section.

3 In the table, enter the following settings:

Expression	Unit	Description
imag(freq)	1/s	

4 In the **Eigenfrequencies (Study 1)** toolbar, click **Evaluate**.

EIGENFREQUENCIES (STUDY 1)

1 Go to the **Eigenfrequencies (Study 1)** window.

2 Click **Table Graph** in the window toolbar.

RESULTS

Table Graph 1

1 In the **Model Builder** window, under **Results>1D Plot Group 2** click **Table Graph 1**.

2 In the **Settings** window for **Table Graph**, locate the **Data** section.

3 From the **x-axis data** list, choose **Eigenfrequency (Hz)**.

4 From the **Plot columns** list, choose **Manual**.

5 In the **Columns** list, select **imag(freq) (1/s)**.

6 Locate the **Coloring and Style** section. Find the **Line style** subsection. From the **Line** list, choose **None**.

7 From the **Color** list, choose **Red**.

8 From the **Width** list, choose **2**.

9 Find the **Line markers** subsection. From the **Marker** list, choose **Asterisk**.

10 Click to expand the **Legends** section. Select the **Show legends** check box.

11 From the **Legends** list, choose **Manual**.

12 In the table, enter the following settings:

Legends
Fixed linearization point (100 Hz)

Table Graph 2

1 In the **Model Builder** window, right-click **1D Plot Group 2** and choose **Table Graph**.

2 In the **Settings** window for **Table Graph**, locate the **Data** section.

3 From the **x-axis data** list, choose **real(freq) (Hz)**.

4 From the **Plot columns** list, choose **Manual**.

5 In the **Columns** list, select **imag(freq) (Hz)**.
6 Locate the **Coloring and Style** section. Find the **Line style** subsection. From the **Line** list, choose **None**.
7 From the **Color** list, choose **Green**.
8 From the **Width** list, choose **2**.
9 Find the **Line markers** subsection. From the **Marker** list, choose **Circle**.
10 Locate the **Legends** section. Select the **Show legends** check box.
11 From the **Legends** list, choose **Manual**.
12 In the table, enter the following settings:

Legends
`Equally spaced linearization points`

13 Right-click **Table Graph 2** and choose **Duplicate**.

Table Graph 3

1 In the **Model Builder** window, click **Table Graph 3**.
2 In the **Settings** window for **Table Graph**, locate the **Data** section.
3 From the **Table** list, choose **Table tableUpdate (update sweep)**.
4 Locate the **Coloring and Style** section. Find the **Line markers** subsection. From the **Marker** list, choose **Diamond**.
5 From the **Color** list, choose **Blue**.
6 Locate the **Legends** section. In the table, enter the following settings:

Legends
`Iterative correction`

Eigenfrequencies

1 In the **Model Builder** window, under **Results** click **1D Plot Group 2**.
2 In the **Settings** window for **1D Plot Group**, type `Eigenfrequencies` in the **Label** text field.
3 Locate the **Plot Settings** section.
4 Select the **y-axis label** check box. In the associated text field, type `imag(freq) (Hz)`.
5 Select the **x-axis label** check box. In the associated text field, type `real(freq) (Hz)`.
6 Locate the **Axis** section. Select the **Manual axis limits** check box.
7 In the **x minimum** text field, type `200`.

8 In the **x maximum** text field, type `1000`.

9 In the **y minimum** text field, type `0`.

10 In the **y maximum** text field, type `450`.

11 Locate the **Grid** section. Select the **Manual spacing** check box.

12 In the **x spacing** text field, type `50`.

13 In the **y spacing** text field, type `50`.

14 Locate the **Legend** section. From the **Position** list, choose **Upper left**.

15 In the **Eigenfrequencies** toolbar, click **Plot**.

Mode Shape (solid)

In the **Model Builder** window, right-click **Mode Shape (solid)** and choose **Duplicate**.

Mode Shape (solid) 1

1 In the **Model Builder** window, click **Mode Shape (solid) 1**.

2 In the **Settings** window for **3D Plot Group**, locate the **Data** section.

3 From the **Dataset** list, choose **Study Eigenfrequency Update (autogenerated)/ Parametric Solutions 1 (sol3)**.

4 From the **Parameter value (iFreqUpdate)** list, choose **1**.

5 In the **Mode Shape (solid) 1** toolbar, click **Plot**.