CEGM2008 - Model Updating Case Study

This instruction is written to ease the completion of the provided Python templates for the case study. As a prerequisite for successfully completing the scripts it is assumed that the reader has completed the workshops and has identified the optimization variable(s) and formulated a cost function.

1 Code structure and dependencies

The goal of this case study is to update the FE model by changing two model parameters, such that the modes and natural frequencies of the FE model match the modes and natural frequencies that were identified. The similarity between the calculated and measured data is quantified by the objective/cost function. This function should be zero if the data from both sources is identical and positive if this is not the case. Note that the optimization parameter is the only variable in this function. Hence, matching the calculated and measured data translates to finding the value of the optimization parameter for which the cost function is minimized.

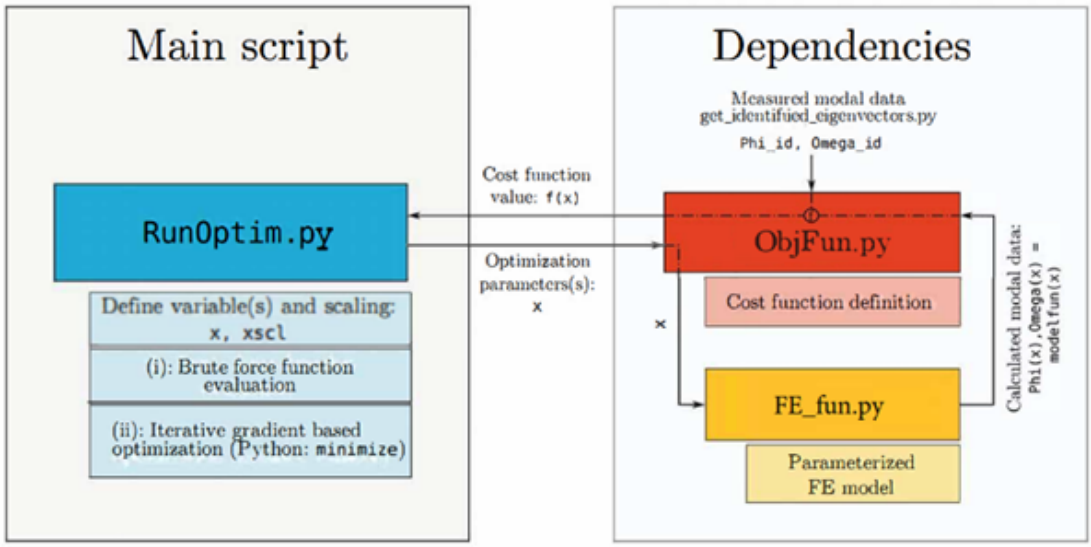

Figure 1: Code structure and dependencies

The global structure of the code is illustrated in Figure 1, and will be explained briefly in this section. The main function to run is: RunOptim.py. This function will be used to minimize the cost function w.r.t. the model parameter. The cost function is defined in ObjFun.py, and is dependent on the optimization parameter. In order to compute the objective function, ObjFun.py needs the measured modal data as well as the calculated modal data from the FE model for each

optimization parameter within the defined range. To obtain the latter, ObjFun.py calls FE-fun.py, which is a parameterized FE model giving the desired output (modal characteristics) as a function of the optimization parameter.

RunOptim.py determines the optimal parameter(s) in two different ways: (i) by performing a brute force assessment of the objective function, and (ii), by using a local gradient based optimization algorithm. For the brute force assessment the cost function will be evaluated for a vector of predefined potential parameter values. The parameter resulting in the smallest cost function value is selected as optimal calibration parameter.

The gradient based optimization algorithm iteratively determines an optimal parameter. Numerically computed gradient information of the objective function is used to determine a new parameter value which results in a potentially lower objective function value. The algorithm iterates until the optimality criteria are met. These criteria typically includes a first-order optimality criterium, which states that the iterations stop if the gradient of the objective function w.r.t. the parameter is zero. In order to start the iterative procedure an initial guess of the optimal parameter value has to be provided.

2 Python Instructions

1. Open RunOptim.py. This is the main function file, where the optimization variable(s) is defined, a brute force assessment of the objective function is performed, the iterative optimization toolbox is called, and visual output is generated.

   (a) Define the bounds of the optimization variable as well as the number of intermediate values: ($x1 = \star * \star$). Also provide the name of the variable, which will be used for plotting ($x1txt = \star \star *$).

   (b) Scale the variables by assigning a scale factor to sclx1. You can think of normalising x1 with its maximum value: $\mathrm{scl}\, x1 = \max(\mathrm{x1})$. The new vector x1, holding the scaled variable, will be used to evaluate the objective function in a brute force fashion. The iterative gradient-based optimization algorithm will use the first and last entry of $x1$ in order to confine the solution space. See section 3 for a note on the importance of scaling the optimization variable.

   (c) Provide an initial guess of the optimum: $x0$. This is required to start the iterative optimization procedure.

2. Open ObjFun.py. This function file holds the objective function definition and is called by RunOptim.py

   (a) Define the cost function. The code is prepared such that contributions to the cost function per identified frequency/mode can be formulated individually. Upon completing the loop the contributions have to be summed.

3. Open FE-fun.py. This function file contains a parameterized FE model. The FE model is identical to the model used in previous practicals. The input to 'FE-fun.py' is the optimization variable. The output consists of the model eigenfrequencies and corresponding mode shapes.

   (a) Assign the optimization variable to a variable in the FE model such that the function output is dependent on the function input.

3 Comments on scaling of variable

Scaling of the optimization variable(s) is of importance for gradient based methods. Consider an objective function whose value increases from $f(x_1) = 0.34$ to $f(x_2) = 0.35$, by changing the E-modulus of steel from $x_1 = 210E9$ [Gpa] to $x_2 = 220E9$ [Gpa]. A finite difference approximation of the derivative equals: $\frac{f(x_2)-f(x_1)}{x_2-x_1} = \frac{0.01}{1e10} = 1e-12$, which is smaller than the default tolerance of the first order optimality criterium in the optimization toolbox, and will therefore numerically resemble zero, i.e. a local minimum! Now consider the case in which the derivative is approximated using scaled optimization parameters, e.g. by normalising both E moduli values by $220E9$. Repeating the computations results in a finite difference approximation of $\frac{0.01}{((220E9-210E9)/220E9)} = 0.22$, which is significantly larger and will not be considered as a local minimum. This example illustrates why scaling of the variables is important before solving the problem using the optimization toolbox. Note that in order to obtain correct cost function values, the variables need to be scaled back in the cost function definition. The scaling procedure is implemented in the main scripts and its dependencies, and requires no further modifications besides the specification of the desired scale factor.

4 Summary: required completions

1. RunOptim.py:	Define parameter bounds, spacing, scale factor and the name of the optimization variable.
2. RunOptim.py:	Provide an initial guess of the optimum value for the gradient based optimization.
3. ObjFun.py:	Define the cost function.
4. FE-fun.py:	Assign the function input to a variable in the script.