MDO assignment; v. 1.1

John T. Hwang, Justin S. Gray, John P. Jasa, and Joaquim R. R. A. Martins

February 27, 2017

For all problems in this assignment, we will use the `problems.py` file. By supplying command-line arguments to the file, you can choose which problem to run. For example, `python problems.py prob1` runs the code for problem 1. Available options are `prob1, prob2, prob3ab, prob3c`. Sections to be modified for each problem are labeled with a comment block.

1. **Structural optimization.**

   (a) This script performs structural analysis and optimization of a tubular beam clamped in the middle. Run the optimization, first with uniform loading and then again with tip loads applied. You must change the code where we set the `loads` variable to use tip loads. What optimized thickness distributions do you see for each case?
   Commands:

      i. run the optimization: `python problems.py prob1`
      ii. view the results: `python plot_all.py s`
      iii. view the optimization history: `python OptView.py s`

   (b) Run the optimization with tip loads applied for a range of different mesh sizes by changing the `num_y` value. Plot the computation time vs `num_y`.

   (c) The script produces an html file, `prob1.html`, that can be useful for studying the problem structure. You can open this file in any web browser. What is the physical interpretation of this problem? Examine the code to find what is actually being optimized. That is, what are we minimizing and subject to what constraint?

2. **Multidisciplinary analysis.** Couple aerodynamics and structures together.

   (a) Open `aerostruct.html` to use a guide. Assemble the aerostructural analysis group following the layout presented there. For this problem run with `python problems.py prob2`. A few components have been placed for you, but you must add the remaining components to the correct groups. Once you have them set up correctly, you can run the problem and view the results using `python OptView.py as` and `python plot_all.py as`. Here we are only performing analysis, not optimization, so only one iteration will be shown in each visualization tool.

   (b) Now that you have assembled the analysis groups correctly in part (a), we will try different nonlinear solvers. Look at this OpenMDAO documentation page to see what solvers are available and how to use them. Note that we are focusing on nonlinear solvers here and will examine linear solvers in part (c).

   Try `nl_gauss_seidel`, `newton`, and `hybrid gs_newton` for different `num_y` values and then compare run times with these different solvers. Note that the hybrid solver is defined within the OpenAeroStruct directory, not in the OpenMDAO solvers page. Examine the file `gs_newton.py` for its details. Now try to run the problem with the Newton solver in the `root` group instead of the `coupled` group. Why do we normally put the nonlinear solver on the `coupled` group instead of the `root` group?

(c) Again visit the OpenMDAO documentation page to see what linear solvers are available. Try `ln_gauss_seidel`, `scipy_gmres`, `scipy_gmres` with a preconditioner, and `ln_direct` solvers while using the Newton nonlinear solver. Which ones can successfully converge the linear problem? Which one gives the fastest convergence for the Newton solver? Why should we not we use the `DirectSolver` with high-fidelity problems?

3. **Multidisciplinary optimization.** Now that you've set up the aerostructural analysis, you're ready to try aerostructural optimization.

   (a) Compute the semi-analytic derivatives of the multidisciplinary system by running `python problems.py prob3ab`. Take note of the run times and derivatives values output by the run script. You can visualize the aerostructural system by running `python plot_all.py as`. Do the derivative values physically make sense?

   (b) Compute the same derivatives using finite differences and compare the timings for different `num_y` values. Use the same code, but add a few lines to force the system to use finite-differencing to compute the derivatives. Try different step sizes to see how the results change. Examine this OpenMDAO documentation page to see how to use finite-differencing.

   (c) We will now perform aerostructural optimization. Add the following design variables in the appropriate locations:

   - 'twist', lower = -10, upper = 10
   - 'alpha', lower = -10, upper = 10
   - 'thickness', lower = 0.003, upper = 0.25, scaler = 1000

   and the follow objective and two constraints respectively:

   - 'fuelburn', scaler=1e-3
   - 'failure', upper = 0
   - 'eq_con', equals = 0

   Now run the code using `python problems.py prob3c`. This optimization will take some time, but you can monitor the progress while it runs. Without stopping the optimization, open a second command window and type the command:

   `python OptView.py as`.

   You can change the settings to adjust what variables you're plotting and you can check the `Automatically refresh` option at the bottom of the checkbox list to have OptView update the plots as new iterations are saved.

   You can also open a 3D visualization of your wing by typing the command:

   `python plot_all.py as`

   (d) Now that you've run the aerostructural optimization case, experiment with different solver parameters and design variable options to find the optimum in the shortest run time. You must keep the same design variables, objective, and constraints as in part (c), but you are free to vary the solver setup, parameter scaling, and optimizer settings. Refer to the previous OpenMDAO documentation pages for more information about possible options.

   You must use `num_y = 11` and your final fuelburn measure must be below 67384 kg for the optimization to be considered successful. The top five groups who use the fewest function evaluations will receive extra credit. 10 points for number 1, 8 points for number 2, and so on.

Notes

- We use a Kreisselmeier-Steinhauser (KS) function to aggregate the multiple stress constraints into a single constraint. Because this is a conservative aggregation, some of the structural elements may be well below the failure limit instead of right at the limit.
- When visualizing results using `OptView.py`, use the `Show 'major' iterations` checkbox to filter out the linesearch results to only view the major iterations.
- The color scheme of the spar in the `plot_all.py` script corresponds to the thickness of the spar.
- For the twist and thickness design variables, we directly control the twist angle of each mesh point and the thickness of individual FEM elements. In a full-scale optimization we would create a smoothing parameter so individual elements or panels could not vary greatly compared to their neighbors. You may notice the optimizer exploiting the ability to control individual panels as you use larger `num_y` values.
- OpenAeroStruct has analytic derivatives computed for most components, but not all of them. The following components **do not** have analytic derivatives:
  - `SpatialBeamFailureKS`
  - `SpatialBeamFEM`
  - `SpatialBeamWeight`
  - `SpatialBeamVonMisesTube`
  - `VLMGeometry`
  - `VLMCirculations`

  Because we use complex-step over these internal components, it may be less computationally expensive to use finite-differencing over the entire system instead of this semi-analytic method. While the entire model has relatively few inputs and outputs, internal components are passing larger arrays (e.g. mesh objects, section forces), which necessitate more computations to obtain their partial derivatives. In general for larger systems, analytic derivatives are the only proper way to perform optimization because they scale much better than finite-differencing and are more accurate.
- If your optimization is taking too long to run, try using a smaller `num_y` value to test different ideas. Note that works well for some values of `num_y` may not work well for others.