IMPLEMENTATION OF A VORTEX LATTICE METHOD IN A HETEROGENEOUS PROGRAMMING LANGUAGE ENVIRONMENT.

Tomas Melin*, Alessandro Augusto Gastaldi **, Mengmeng Zhang **
*** - Svenska Flygtekniska Institutet, ** Airinnova**

Keywords: *Aerodynamics, conceptual design, method implementation, AGILE*

Abstract

The development of new workflows in aircraft design has created a need to re-implement legacy analysis software to fit in modern frameworks. Older software architectures do not necessarily enable the full potential of modern hardware and software environments. European research in the AGILE project has as a goal to enable a software framework for innovative collaboration between heterogeneous teams of experts, using various in-house tools, specifically in different programming language environments and platforms.

This paper describes the work done to shift the Tornado vortex lattice method, (VLM) from a Matlab-centric implementation to a platform-independent implementation.

1 Background

The AGILE project is aimed at developing and testing multidisciplinary optimization using distributed analysis frameworks. An ultimate goal is to prove a 40% speedup of solving realistic MDO problems when compared with the state-of-the-art. The project is funded under Horizon 2020 and runs from 2015 to 2018 [1]. The overall methodology to tackle the workflow challenges in a distributed design environment has been described in detail by Prakasha et al [2], Lefebvre et al. [3] and Nagel et al. [4]. A previous EU-funded research project, SimSAC, produced the software environment CEASIOM (Computerized Environment for Aircraft Synthesis and Integrated Optimization Methods) was developed at CFS Engineering [5].

At Airinnova, one of the tools used in research and design is the vortex lattice method (VLM) Tornado, which is implemented in Matlab. The software is currently under development at the Swedish aeronautical institute – Svenska Flygtekniska Institutet.

For the purposes of the AGILE project, the Tornado code was refactored to more modern architecture to fit in the AGILE workflow. The Matlab implementation, while validated, mature and very popular, has two peculiarities: Firstly, while the Tornado code itself is freely distributed under the GNU-GPL license, its intrinsic link to the commercially licensed software Matlab limits the use of the software. Secondly, as Matlab is an interpreted software, computational speed is not a primary objective.

The work done to port the Tornado code was shifting core computational routines into a compiled executable coded in C, wrapped in a user interface coded in Python. The Python interface, developed at Airinnova, was intended to be used both for the Tornado code as well as for a stand-alone Double Lattice code.

2 Code architecture

The information flow architecture is shown in Figure 1. The Python segment covers the user interface, pre- and post-processing, while the C executable perform the computationally heavy operation. This approach was selected to enable both fast computations and a non-commercial license environment.

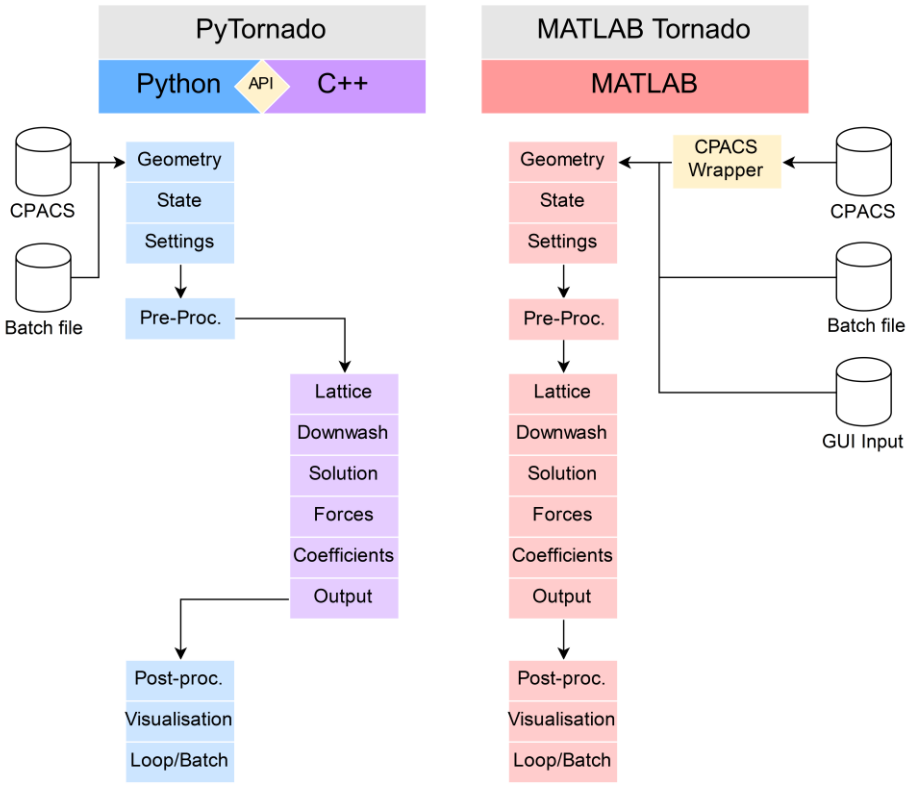


Fig. 1. Program workflow. The PyTornado acts as a wrapper for the compiled C executable.

Alternatively, Python can be used to batch script computations or directly link the code into the AGILE workflow through the remote component environment (RCE) using the Common Parametric Aircraft Configuration Schema (CPACS) as an aircraft descriptor. The compiled C code executes the actual numerical simulation. In parallel, the core functions are callable from the Matlab implementation. Technically, the C-code can be run stand-alone if no user interface is required.

In parallel, the information flow in the legacy Matlab implementation is also shown in Figure 1. For debugging purposes, there is added functionality in the Matlab code to call the C executable through a batch input file interface.

2.1 Data Flow

The flow of information within the PyTornado implementation is as follows. First, three types of input are required:

- **Geometry**: The geometry of the wing or aircraft to be simulated needs to be made available to the code in a readable format, allowing efficient discretization.
- **Flight State:** Multiple sets of operating conditions may be defined for batch computations. These consist primarily of angle of attack, velocity, altitude and control surface deflection.
- **Solver parameters**: Finally, it remains to define those settings relating to the method of analysis: The type of mesh, wake type, correction factors etc. This includes any parameters not directly related to either state or geometry.

In the current implementation, this can be done either by 1) using the user interface and entering the geometrical parameters at runtime 2) a script file in batch mode or 3) an XML file from the AGILE workflow.

Once all the necessary inputs are available, the computation code will perform the following steps in order:

- **Planform geometry:** Generating the *xyz*-coordinates of the corner points of quadrilateral partitions that make up each wing (more generally, each lifting surface) and control surface – see Figure 2. This can be done either in Python, or in C directly.
- **Discretization:** Calculating coordinates of the panels, vortices and collocation points for VLM.
- **Downwash coefficients:** Computing the matrix of AICs (aerodynamic influence coefficients) and right-hand-side terms.
- **Solution:** Solving the linear system of equations for vortex strength using efficient linear algebra routines.
- **Forces:** Calculating the aerodynamic forces from known vortex strengths and aerodynamic influence coefficients.
- **Coefficients:** Integrating forces and moments and normalizing the result to obtain the non-dimensional aerodynamic coefficients.
- **Output:** Formatting the data to suit the post-processing functions. This too is possible done in both the C and Python environments.

Once the core analysis routines have been completed, post-processing functionality such as case comparisons and plotting are delegated to the Python environment.

2.2 Geometry Definition

Along with the changes brought to the program architecture, the rewriting of Tornado in Python provided an opportunity to re-think its core data structures for more efficient computation while simplifying interaction with external tools.

The CPACS format developed at DLR [4] has proven to be instrumental in the communication and interfacing of data between partners in the AGILE project. For this reason, it was a natural choice to use a similar hierarchical structure for the internal geometry definition of PyTornado.

The aircraft planform is defined as an assembly of lifting surfaces, each of which is composed of multiple quadrilateral segments. This allows for the modeling of complex wings with varying twist, taper, camber and dihedral distributions, with elements that are straightforward to subdivide into vortex panels. The lifting surface segments are defined by the following properties:

- Span
- Chord (inboard and outboard)
- Twist (inboard and outboard)
- Sweep
- Dihedral
- Airfoil geometry

The geometrical parameters are further shown in Figure 2.

The lifting surface tracks the span-wise position of these segments. This allows for the definition of control surfaces in terms of their relative span- and chord-wise location, as in the CPACS schema. The deflection angle can then also be provided for each control surface.

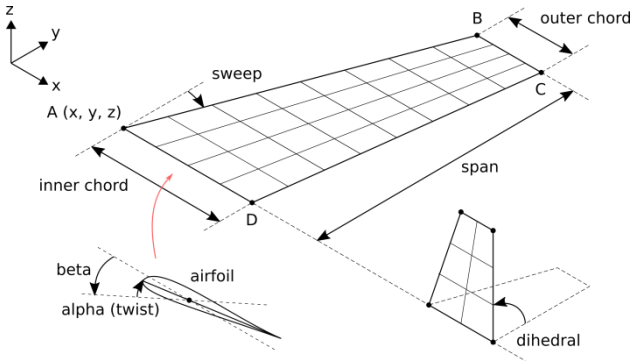
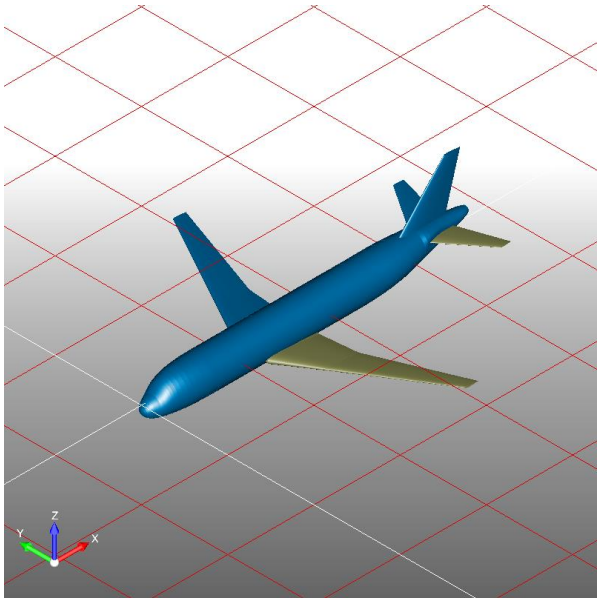


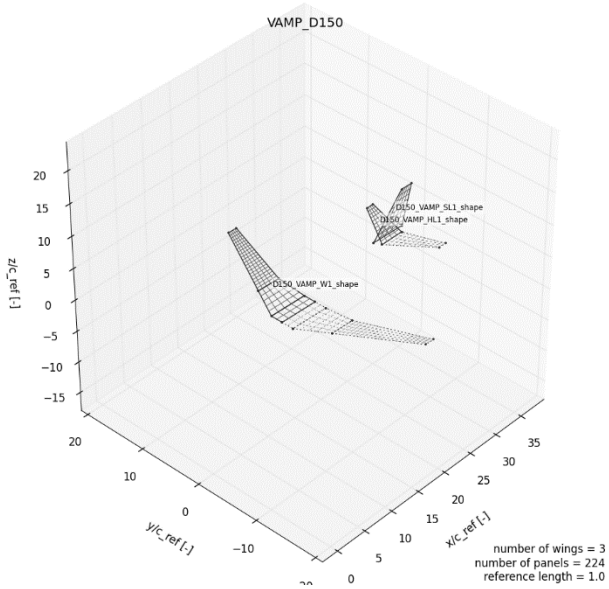
Fig. 2. Geometry definitions.

The above hierarchical structure is implemented in Python, simplifying pre- and post-processing operations on individual geometry components. The TIXI library, developed at DLR, is used to interact with CPACS as an input format. The planform geometry is extracted by computing the average plane of CPACS wings and wing segments, which correspond one-to-one with the PyTornado lifting surfaces and segments. Then, airfoil and control surface information is loaded and processed.

When the complete geometry is generated, the segment corner point coordinates are shared in-memory with the grid generation routines in C, along with any discretization parameters. The resulting panel corners, vortex, and collocation point coordinates are stored in contiguous arrays for performance and memory efficiency. Airfoil data is used to shift the panel normal vectors by the camber distribution. Control surfaces are treated separately in order to ensure a conformal and sufficiently fine discretization. Figure 3 show the TIXI/TIGL representation of the D150 geometry as rendered from an AGILE hangar CPACS file. Figure 4 show the corresponding potential flow computational lattice.



Fig, 3. The AGILE D150 geometry.



Fig, 3. The D150 potential flow computational mesh.

Much of the CPACS and PyTornado geometry definition were already aligned with the old Tornado standard.

2.3 Tornado development

During the work with porting the Tornado code, some new functions were added to enhance the modeling capability of the software:

- Control surface taper
- Slats
- Symmetric control surface separation
- Control surface text tags
- Geometry database text tags
    - Aircraft geometry name
    - Project name
    - Project engineer

Control surface taper is a function that enables the user to define different taper ratios for trailing edge control surfaces and the corresponding main wing element. This allows to define, for example, rectangular control surfaces on a tapered wing.

Slats are simply control surfaces hinged forward of the main wing element. These too can have different taper to the main wing element. Built-in logic should prevent collision between the leading and trailing control surface. The main aerodynamic effect of slats – to prevent leading edge flow separation at high angles of attack – isn't modelled, as separation isn't captured by potential flow. It will, however, model the aerodynamic effects of the change in camber.

In the old Tornado standard, the trailing edge control effectors on a symmetric wing were always connected through the same control channel. After requests from flight mechanics engineers and control system designers, the effectors were decoupled and given their own channel. In addition, a string field was added to facilitate naming each control effector.

Similarly, in the geometry file, a header with text fields for geometry name, project name and the project engineer were added for data traceability.

3 Method

3.1 Debugging and verification

The debugging of the core Tornado functions was enabled through the Matlab-C interface. The code was continuously verified during development. Each component in the workflow was individually evaluated for integrity. This approach was made possible as the old Matlab code is available and well-validated.

When the C code was finalized, two different verification acceptance tests were applied. The verification cases for the old Matlab version were re-run in both environments to test overall accuracy and a Monte-Carlo approach with random geometries were tested in both environments to ensure converged behavior.

For comparisons of computational speed, a rectangular wing with aspect ratio 6 was evaluated at 5 degrees angle of attack several times, with increasing panel density.

4 Results

Results have been promising with a tenfold speed increase in comparison to the original implementation when calling the C-functions instead of the Matlab Tornado core. The results for different mesh sizes are shown in Figure 4. With mesh sizes larger than 500 panels, execution times of both the Matlab and C-implementation grow at the same rate. Interestingly, at around 70 panels – which is not an unusual panel count – both implementations are equal in time consumption.

6 Conclusion

The project has shown that it is feasible to port legacy software to a platform independent framework. Furthermore, it has demonstrated the use of the same C-code from both a Python platform and a Matlab platform.

7 Discussion

In the porting to C, no extra work was spent on optimizing the code for best memory usage. A higher computational speed of the C-code could therefore be expected in future, optimized, versions.

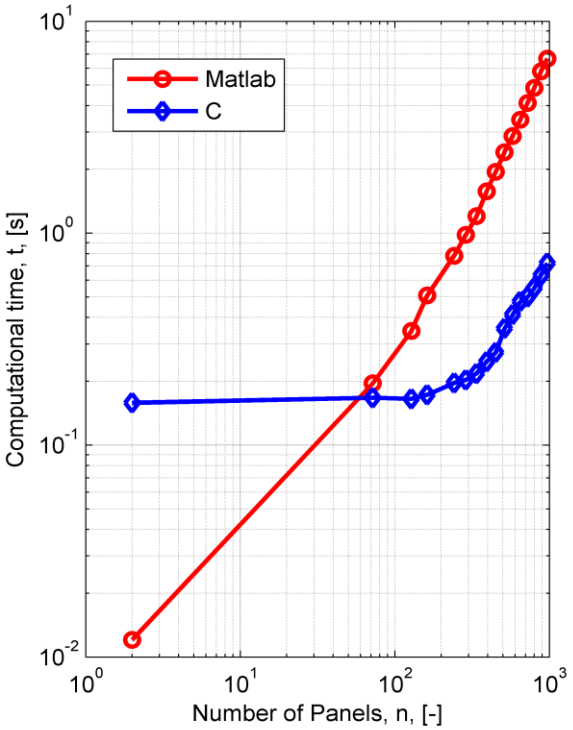


Fig. 4. *Code speedup comparison: Matlab vs. C*

7.1 Lessons learned

- While a VLM tool is very suitable for design automation due to its low geometrical overhead, meshing and grid convergence studies still require some manual intervention with lattice density and distribution to assure good results.
- A software project as large as AGILE would have benefitted greatly from a common version/revision control system for software, essentially an AGILE project for software development.
- With three software environments to maintain, further code changes will take some more time to implement and verify cross-platform.

7.2 Future Work

During the continued development work of the Tornado code some key future development items were identified, but not directly addressed within the scope of this paper.

- Storing wing profile definition together with the rest of the geometry data.
- Harmonization towards the CPACS standard, e.g. changing the name of the geometry struct to “wings”.
- Functionality for baseline comparisons.
- Enabling better connection between results and geometries when archiving project data
- Further investigation of distributing the software functionality as a service rather than as distributed code.

The PyTornado suite is still being developed; any comments or functionality requests from potential users would be greatly appreciated by the authors.

References

[1] Anon, *AGILE Project ID: 636202*, CORDIS, 2018

[2] Prakasha P.S. Ciampa P.D. Nagel B. Boggero L. and Fioriti M. *Collaborative Systems Driven Aircraft Configuration Design Optimization*, Proceedings of ICAS2016.

[3] Lefebvre T., Bartoli N., Dubreuil S., Panzeri M., Lombardi R., D'Ippolito R., Vecchia P.D., Nicolosi F., and Ciampa P.D. . *"Methodological enhancements in MDO process investigated in the AGILE European project"*, 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference, AIAA AVIATION Forum, (AIAA 2017-4140). doi.org/10.2514/6.2017-4140

[4] Nagel, B., Böhnke, D., Gollnick, V., Schmollgruber, P., Rizzi, A., Rocca, G.L. and Alonso, J. (2012) *“Communication in aircraft design: can we establish a common language?”*, 28th International Congress of the Aeronautical Sciences, ICAS 2012, Brisbane, Australia.

[5] CFS Engineering SA. *CEASIOM - Computerised Environment for Aircraft Synthesis and Integrated Optimisation Methods.* www.ceasiom.com/. Accessed June 29th, 2018.

8 Contact Author Address

Dr. Tomas Melin.
Svenska Flygtekniska Institutet.
Westmansgatan 37A, 58216 Linköping,
Sweden

Tel: +46(0)709 632 698
melin@sftiab.se

Copyright Statement