Modernization of a Vortex-Lattice Method with Aircraft Design Applications

by

Tyler J. Souders

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2021 by the
Graduate Supervisory Committee:

Timothy Takahashi, Co-Chair
Marcus Herrmann, Co-Chair
Werner J.A. Dahm

ARIZONA STATE UNIVERSITY

May 2021

ABSTRACT

The Vortex-lattice method has been utilized throughout history to both design and analyze the aerodynamic performance characteristics of flight vehicles. There are numerous different programs utilizing this method, each of which has its own set of assumptions and performance limitations. This thesis highlights VORLAX, one such solver, and details its historic and modernized performance characteristics through a series of code improvements and optimizations. With VORLAX, rapid synthesis and verification of aircraft performance data related to wing pressure distributions, stability and control, and Federal Regulation compliance can be quickly and accurately obtained. As such, VORLAX represents a class of efficient yet largely forgotten computational techniques that allow users to explore numerous design solutions in a fraction of the time that would be needed to use more complex, full-fledged engineering tools. In the age of modern computers, one hypothesis is that VORLAX and similar "lean" computational fluid dynamics (CFD) solvers have preferential performance characteristics relative to expensive, volume grid CFD suites, such as ANSYS Fluent. By utilizing these types of programs, tasks such as pre- and post-processing become trivially simple with basic scripting languages such as Visual Basic for Applications or Python. Thus, lean engineering programs and methodologies deserve their place in modern engineering, despite their wrongfully decreasing prevalence.

DEDICATION

I would like to dedicate this work to my family and to my girlfriend, Lily Cothren. Without their combined support, I would not be where I am today.

ACKNOWLEDGMENTS

I would like to thank Professor Takahashi for always supporting me in my ventures are pushing me to be the best I can be academically, as well as for his advice regarding careers and academia.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

VORLAX is a computer program initially developed by Luis R. Miranda, Robert D. Elliot, and William M. Baker of Lockheed California in the late 1970's [1]. Originally intended to run on machines such as the IBM 360, the code follows the typical conventions seen with computational fluid dynamics (CFD) programs of the time. Namely, it was developed on a punch card system, relying on tape decks for "memory" read and write operations. As such, the program relies solely on single dimensional arrays, runs in single precision, and partakes in aggressive saving of variables to simplify calculations. For instance, it saves trigonometric projections for later use as opposed to calculating them each time. This diligent memory and CPU management was necessary for the program to operate in its original era; however it exists as a perk in the year 2021.

To understand the appeal of *VORLAX*, specifically relative to other modern vortex-lattice codes such as TORNADO [7], it is necessary to look at the computational limitations imposed by the IBM 370 and compare it to modern technologies. When

Table 1: Comparisons to Modern Personal Computers and Devices [2] [3] [4] [5] [6]

Device	**CPU Clock Speed (MHz)**	**RAM (MB)**
IBM 370	8.7	8
Raspberry Pi Zero	1000	512
Apple iPhone 12	1800	4000
Macbook Air M1 (Base)	3200	8000
Lenovo Thinkpad P-Series	5100	32000

faced with comparisons to modern technology, there are orders of magnitude in difference. One of the most basic non-microcontroller personal computing chips on sale now is the Raspberry Pi Zero, which boasts a relatively massive 1.0GHz processor with 512MB of RAM relative to the 8.7MHz processor and 8MB of RAM available on a high-end IBM 370. While I am not suggesting that someone would want to use a Raspberry Pi to run *VORLAX*, it is important to see the difference in performance capabilities. Taking a more serious comparison, I chose the ThinkPad P1, which is a standard business laptop featuring an Intel Xeon processor clocked at 2.8GHz which can "boost" to 5.1GHz along with 32GB of RAM. At these scales, the CPU is running at a rate two and a half orders of magnitude greater (without accounting for the parallelization options found on the Intel chip, since *VORLAX* runs sequentially), and the system has 4000 times the amount of memory available. While *VORLAX* exists as a 32-bit compile in order to maximize speed and simplicity, thereby limiting it to 2000GB of heap memory on Windows systems, this remains a massive difference relative to the IBM system. Thus, *VORLAX* exists as a highly optimized, lightweight CFD solver that can run remarkably quickly on modern systems.

VORLAX runs quickly, but it is necessary to outline the operation process in order to appreciate its speed relative to more complicated solvers. CFD suites such as ANSYS Fluent will offer more "accurate" results for a complex flow, however the returns are diminishing with first-order design problems. Furthermore, this improvement comes at the cost of hours upon hours of work to generate models. Even so, unless the engineer spends hours to diligently model each individual rivet, panel junction, window, door, etc. then the "perfect" CFD model has an empirical "crud" drag imposed on top – thereby bringing into question the validity of this "perfect" model [8]. Thus, *VORLAX* offers a unique and efficient way to have "90% accurate" models that can

be pre-processed, run, and post-processed in the amount of time it takes for ANSYS to load on a PC [9].

A large advantage of *VORLAX* relative to more complex methods is that the interfacing system with *VORLAX* is greatly simplified relative to other software suites, thereby existing as a system that is not only faster and much simpler, but also completely avoids reliance on any proprietary mesh generators of post-processing utilities. *VORLAX* operates entirely on the basis of standard UTF-8 encoded text files. The user will generate file inputs, complete with geometric and flight configuration definitions, and will receive two UTF-8 output files. Of these files, one returns basic results regarding quantities such as the lift, drag, and stability derivatives, while the other offers precise pressure distribution information at each control point of the model.

Fortunately, working with these files is remarkably simple. A common tool for reading and writing UTF-8 files following FORTRAN fixed-width column printing is Visual Basic for Applications (VBA). VBA is a great utility for this process because it is included as an underlying utility in the Microsoft Office Suite. Thus, it is possible to automate the reading and writing of these data files using a method that an employer or educator almost certainly already has access to. With VBA in conjunction with Microsoft Excel, one may construct a suite that shows a geometric preview of the model, writes an input file, runs *VORLAX*, then parses the output to return any desired outputs (these will be more rigorously detailed in Chapter 3).

In the almost unimaginable case where the user does not have access to Microsoft Office products, the interfacing with *VORLAX* follows such well-documented and universal methods that there exist a plethora of tools and libraries that allow users to analyze the data. One very viable method for working with the *VORLAX* data files is

Python. Python is completely free, and with it comes three noteworthy libraries which are useful for working with *VORLAX*. *Numpy* is a library that essentially offers almost all of the functionality of MATLAB (less the plotting and controls functionality) without any charge. For the MATLAB plotting environment, *Plotly* and *Matplotlib* are both suitable for generating two-dimensional plots. *Plotly* offers the additional functionality of generating a three-dimensional model complete with pressure contours which may be manipulated to show different viewpoints, all within the user's internet browser.

Thus, *VORLAX* exists as a highly accessible utility for the analysis of aerodynamics. The program is incredibly accessible, and in the subsequent chapters the theory and operation will be detailed, as well as its applications.

CHAPTER 2

VORTEX-LATTICE THEORY AND GEOMETRY CONSTRUCTION

As mentioned previously, *VORLAX* is an implementation of the Vortex-Lattice Method (VLM), the specifics of which is thoroughly outlined in its own theory guide [1]. For completeness, it is key to discuss the method briefly yet completely in this section. The Vortex-Lattice Method exists as an extension of Prandtl's famous "Lifting-Line Theory". The lifting line theory suggests that a wing of finite span may be modeled by a series of superposed bound vortices, where the change in local circulation is proportional to the local lift [10].

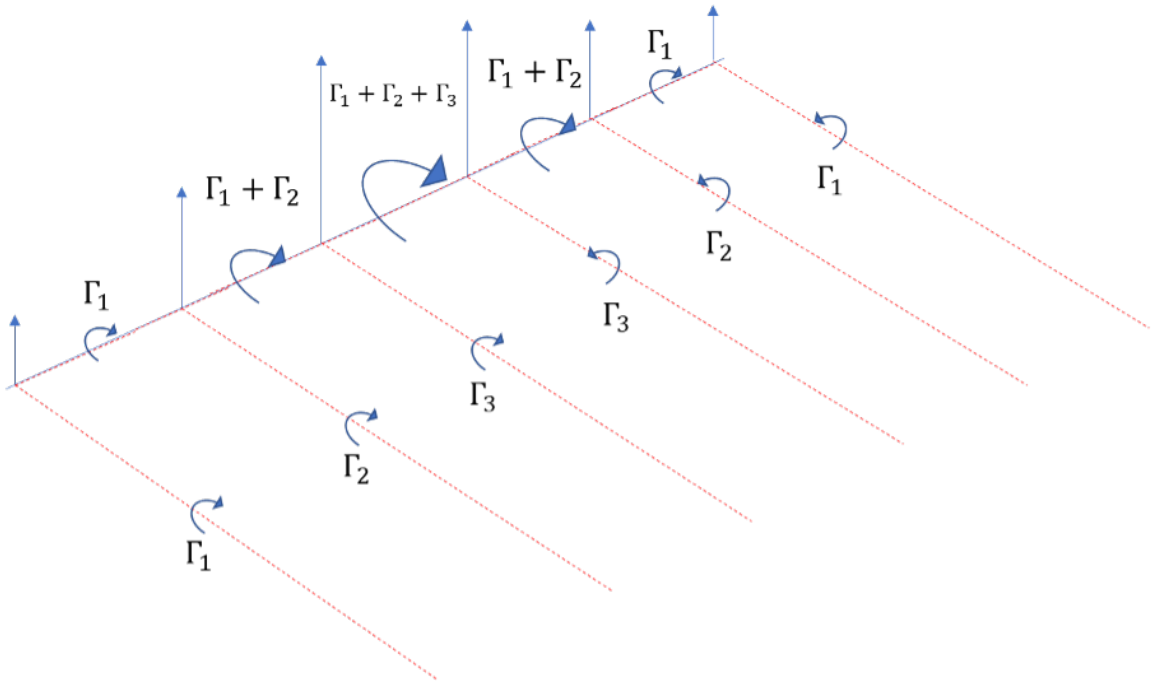


Figure 1: Lifting-Line Theory

By computing this circulation strength, the governing concepts of complex potential

flow are applicable to resolve the pressure distribution along this finite wing. As covered in introductory aerodynamics courses, it is well understood that with a known pressure distribution, simple integration returns the lift, drag, and pitching moment.

The Vortex-Lattice Method fills in where the lifting-line theory begins to fall apart. Lifting-line theory assumes a single, unswept wing with incompressible flow conditions. With any real-world aircraft, these assumptions are completely impractical, as there will always be interference from things such as the fuselage, winglets, flaps, slats, and ailerons. Thus, it is advantageous to employ a more versatile technique, and as such the Vortex-Lattice Method becomes relevant.

Unlike the lifting-line theory, which assumes that the bound vortices are all collinear, the Vortex-Lattice Method lets the use model any geometry as a series of discrete, finite panels. For each of these panels, there is a bound vortex located at the quarter-chord location (which is usually not the same location at the actual aeronautical "quarter-chord" as one would understand for the entire wing, rather it is the quarter chord of each individual panel), as determined to be optimal by [11]. Then, each panel has a "control point" located at the three quarter-chord. The model with consist of hundreds, perhaps thousands of these bound vortex and control point pairs, amounting to a fair, but not obscene, number of discrete points at which the aircraft is analyzed.

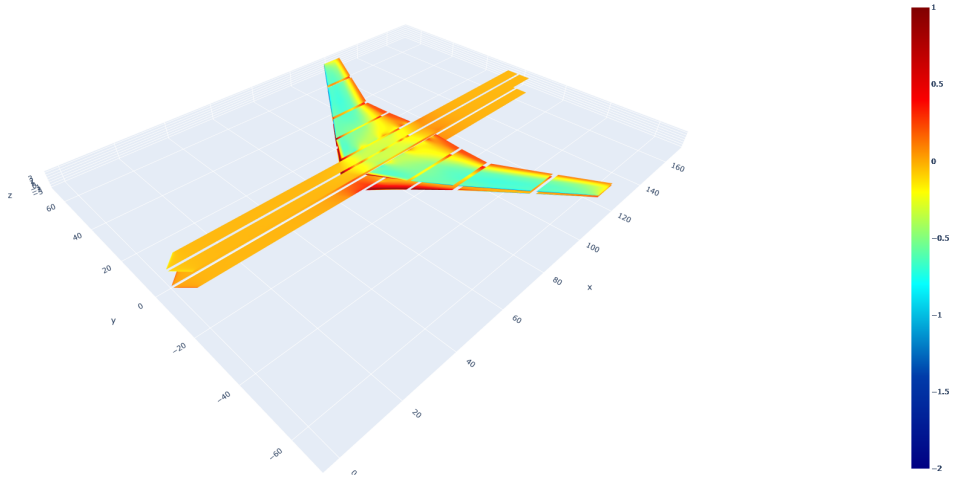


Figure 2: Vortex-Lattice Panel Geometry

$$\text{Total Control Points} = \sum_{k=1}^{NPAN} NVOR_k \cdot RNCV_k$$

For each control point, the surface is assumed to be solid on both sides in the case of a "flat panel" representation or is assumed to be solid on one side in the case of a "sandwich panel" representation. Knowing that each of the bound vortices will impose flow not only at their respective control points, but also everywhere on the model, it becomes a case of linear algebra to solve the complex potential equations to determine the necessary circulation strength, Γ, of each bound vortex in order to impose a zero-flux, Neumann type boundary condition at each control point. However, this becomes computationally limiting, as this means the system of equations involves a dense, non-diagonally dominant, non-positive definite matrix of size $[TotalPoints, TotalPoints]$ that must be solved. Typically, with modern finite-difference based computational fluids solvers, memory management is largely simplified by the usage of sparse, diagonally-dominant matrices [12], which the Vortex-

Lattice Method is unable to take advantage of because of the aforementioned technical limitations. Fortunately, as will be later detailed, the vortex lattice method does not require so many points that this loss in numerical efficiency is limiting – quite conversely, it becomes apparent that *VORLAX* remains so efficient overall that its performance is preferential in comparison to the FDE-based solvers for quick, "back of the napkin" style computations.

However, despite this increase in numerical complexity, the code remains remarkably bulletproof and accurate within its realm of assumptions. These assumptions of the method are that the flow is inviscid, remains attached to the surface, with no shock formation over the wing, and the flow is steady. While the assumption that the flow is inviscid is undeniably inaccurate, this may be remedied by running a flat-plate drag buildup program such as EDET [9][13]. in conjunction with *VORLAX*. These drag buildup methods also exist as efficient FORTRAN codes, so there is little overhead burden to such an addition, and the task of generating geometries in one of the programs based on the input file of another is just as simple and can be easily automated [14]. Regarding shock formation over the wing, it is not viable to use *VORLAX* to resolve these shock effects and the associated pressure and velocity jumps in its present configuration, however the method is perfectly reasonable to use to determine where a shock is likely to form by means of a critical pressure coefficient equation [15].

$$Cp^* = \frac{2}{\gamma M_\infty^2}\left\{\left(\frac{2}{\gamma+1}\right)^{\frac{\gamma}{\gamma-1}}\left(1+\frac{\gamma-1}{2}M_\infty^2(\cos\varphi)^2\right)^{\frac{\gamma}{\gamma-1}}-1\right\} \tag{2.1}$$

Thus, for an aircraft designer whose goal is to ensure the wing experiences shock-free flight conditions, the method is perfectly viable.

While *VORLAX* cannot accurately handle transonic flow pressure distributions, it

is capable of accounting for compressibility effects. To do so, one should understand the *VORLAX* operates by superposing normal-wash and axial-wash components on top of the freestream velocity [1]. It is to these normal- and axial-wash conditions that the well-known Prandtl-Glauert effect is applied. As a result, the pressures are significantly more accurate than those obtained via a blind application of the Prandtl-Glauert scaling factor as suggested in introductory compressible flow texts [16].

To utilize the vortex-lattice code *VORLAX* in aircraft design applications, the user should be well-versed in its capabilities. *VORLAX* may model any connected-flow system as a series of connected finite panels as defined in three-dimensional space by their leading edge coordinates and the chord lengths at their roots and tips. Thus, the user may generate any structure, ranging from a single finite wing to an entire airframe complete with control surfaces and engine nacelles. *VORLAX* offers the functionality to generate a fusiform style structure which is useful for generating cylindrical aerodynamic influence structures such as nacelles. Multiple examples of these structures are presented below:

Figure 3: Single VORLAX Wing

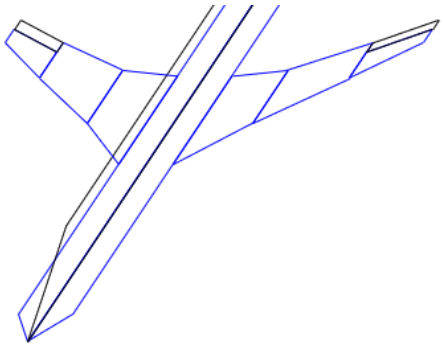


Figure 4: Full VORLAX Panel Aircraft

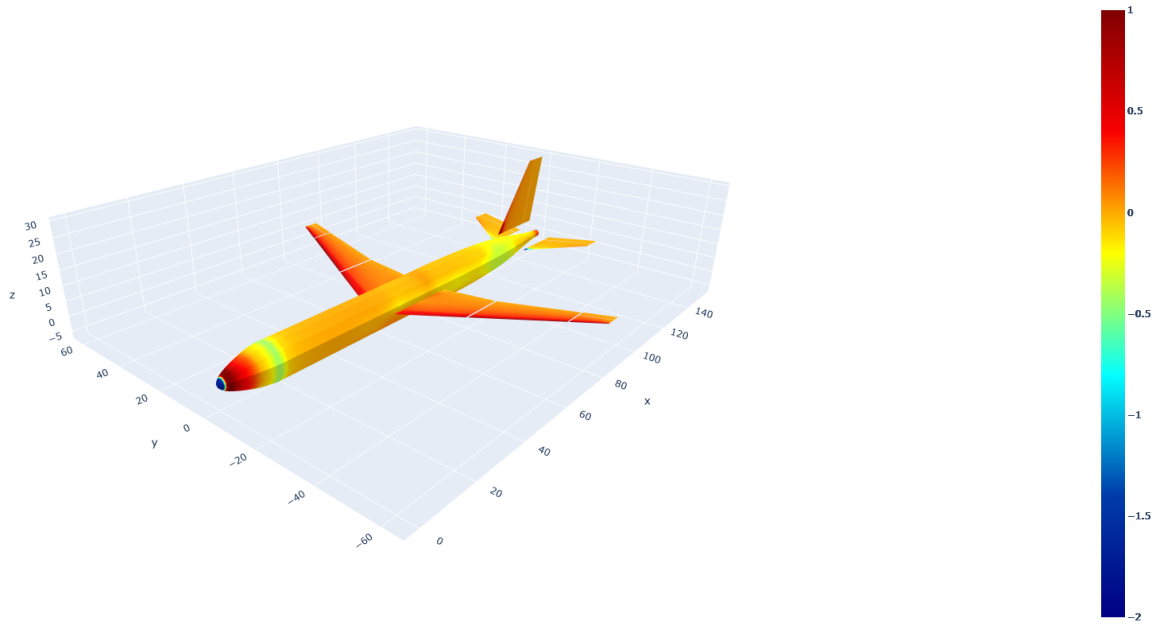


Figure 5: VORLAX Fusiform Representation of a Boeing 757

The versatility of these flat panels should not be under-appreciated. The *VORLAX* code allows the user to define the surface sides which require the zero-flux boundary condition, offering the option of an infinitesimal panel that requires zero-flux on each side as well as the option to only restrict the boundary condition on one single side. By only enforcing the boundary condition on one side, the user may impose structural (panel shape) perturbations in the form of thickness and camber to redefine the aerodynamic shape of the panel, thereby offering more accurate pressure distributions over the surface. When accounting for the thickness and camber, *VORLAX* offers two geometric strategies, the first of which involves leaving the panels as planar structures which changes to the local slope of each panel, which thereby alters the trigonometric normal relation necessary for the zero-flux boundary condition. The other method is to physically relocate the panel to account for these changes in slope and position.

CHAPTER 3

RUNNING VORLAX

As suggested in the previous section, *VORLAX* is not only fast, but it is extremely user-friendly when generating a model and running the program. *VORLAX* was originally written for old-school punch cards, an input format which has now manifested itself as UTF-8 text files. When a user wishes to model an aircraft via *VORLAX*, they will need to provide some basic flight information necessary for the geometry construction and flow computations.

3.1 The VORLAX Header

The user is allowed to define up to 16 individual Mach numbers and angles of attack, thus 256 total configurations in a single run. The user will also define the grid based on whether the control points should be computed linearly (where each row of grid points is equidistant from its neighboring rows) or as a cosine spacing where the rows are grouped at the edges of the panels. Conveniently, *VORLAX* allows the user to mix the spacing methods, allowing for the popular choice of cosine spacing in the chordwise direction (which gives better pressure and induced drag resolution over each chordwise strip), while utilizing linear spacing along the wing which is preferential for pressure resolution near the wing-body junction. One restriction, however, is that these spacing methods are uniform over all of the panels, there is not the option to have linear spanwise spacing for panel (1) while having cosine spacing for panel (2).

The header of the input file also calls for the relaxation parameter used in the

CSOR solver method (further detailed in Chapter 4) and the iteration limit for the iterative solver. The other commonly used feature in the header is the ability to denote a model with sideslip, where the user may denote a sideslip angle for the entire model. Finally, the header will call for the number of panels used to represent the geometry, as well as the location of the center of gravity on the central plane along with the span, mean chord, and reference area of the trapezoidal wing structure. When combined, these parameters allow *VORLAX* to return coefficient parameters such as the lift and drag coefficients (as opposed to dimensional figures).

3.2 VORLAX Geometry Generation

When using *VORLAX* to model an aircraft, the user must supply multiple dimensions to tell the program how the draw each panel. This begins with the user providing four dimensions per panel, the cartesian location of the root point and tip point of the leading edge as well as the length for both the root and tip chord. Following, the user supplies the program the desired number of chordwise control point elements ($RNCV$), as well as the number of spanwise elements ($NVOR$). These two values may differ from panel to panel, giving a total of

$$NPTS = NVOR \cdot RNCV \tag{3.1}$$

per panel. It is desirable to alter these values for each panel in order to obtain a similar grid density on each panel. Thus, in common applications, these numbers will vary significantly. For instance, it is logical to have 50 chordwise points along the length of the fuselage, however that number of points on an aileron will lead to numerical errors due to the proximity of the vortex elements and control points. While having

too sparse of a grid will become problematic as it cannot accurately represent the forces over an entire body, too fine of a grid presents problems because of singularities introduced due to the proximity of elements (where the distance between a vortex element and control point approaches zero, leading to numerical overflow errors).

With the grid spacing, the user will also specify whether the panel needs to account for leading-edge suction effects. This is done via an implementation of Lan's method [17]. One restriction of this method is that it was designed specifically with cosine grid spacing as a consideration. Thus, the computed values for leading-edge suction and induced drag are incorrect when using linear panel spacing. To prevent this bug, which has remained poorly documented until this point, a check was introduced into the *VORLAX* source code. This check will verify that the leading-edge suction flag (SPC) is equal to zero if the chordwise panel spacing is linear and the freestream Mach number is subsonic. Should the check fail, the program is terminated (thereby saving the computation time) and an error message is printed to the output log which informs the user to alter the chordwise spacing flag and re-run the program.

There exists a flag to denote whether a panel is of the fusiform type. If so, the user will provide a list of polar coordinate pairs at a number of chordwise stations to define the shape of the fusiform cylinder. This cylinder will feature the same infinitesimal panels with a vortex element and control point as in other cases, where the number of flat faces is denoted with the $NVOR$ flag. When constructing a fusiform panel, the user must take care to ensure that there are $NVOR + 1$ polar coordinate pairs, as that is how *VORLAX* generates the structure.

Finally, the definition of each panel will ask for the twist angle of each defined chord, along with the indication as to whether the panel is single- or double-impermeable. The user will also provide the number of stations at which the camber is defined,

along with information telling the program whether the panel should be mirrored about the central axis, whether to use the inverse design functionality, or whether the panel is to be of the nonplanar type, where the points are physically displaced to account for camber definitions. The functionality of each of these parameters is thoroughly documented in the original *VORLAX* publication, though it is important to acknowledge their existence for the sake of due diligence.

Regarding the numerous flags, what originally appears to be many features works out to be incredibly simple in practice. In Appendix A, there are several examples of input files, including examples of infinitesimal wings, thick wings, thick and cambered wings, full aircraft with control surfaces, and more. This relatively basic input format offers the user a profound amount of control and returns equally remarkable results.

One benefit of this input system is that it stays the same. When working on the *VORLAX* improvements detailed in Chapter 4, express care was taken to avoid altering any of the input files, as it would break backwards compatibility. The nature of the input files means that complex geometries may be computed with basic knowledge of the aircraft and trigonometric relations. With Microsoft Excel, it becomes possible to rapidly generate the necessary points based on parameters such as the reference area, wing aspect ratio, leading-edge sweep, and taper ratio. Thus, for any proposed change, the program may be altered and run in a matter of seconds, making this option incredibly efficient for aircraft synthesis.

3.3 Parsing VORLAX Outputs

Upon termination of a successful run, *VORLAX* will present the user with a simplified output file which contains basic aerodynamic parameters, as well as a detailed

log file featuring detailed local pressure information over the entire aerodynamic surface. Examples of these output formats are both presented in Appendix A. These output files are generated in a very concrete format, in part due to the history of being a punch card program. Thus, one may develop parsing utilities with confidence that the output formats will not change. During the improvements to *VORLAX* (as described in Chapter 4), care was taken to avoid altering the output file formats. Small changes were completed to improve clarity of the files, such as fixing some of the heading spacing and widening some columns to avoid printing errors, however the functionality was unaltered. When *VORLAX* presents output data, the formatting is consistent between all configurations. This means that it is possible to create utilities that may plot any of a number of solutions for a generic *VORLAX* run. Examples of these utilities are detailed in Chapter 5 alongside discussions of the usage and accuracy of *VORLAX*. Essentially, *VORLAX* generates output information in such a manner that a user can very easily write scripts to generate very detailed graphics and datasets based on nothing other than the data contained within the lone *VORLAX.LOG* output file.

CHAPTER 4

NUMERICAL IMPROVEMENTS TO THE VORLAX CODE

4.1 Operational Changes

The primary motivation for this thesis was to work on the optimization of the *VORLAX* code to greatly improve its utility and reliability. As previously mentioned, *VORLAX* implements a vortex-lattice method to solve for the flow distribution over an aerodynamic surface. The hypothesis was that if *VORLAX* was improved to utilize modern PC RAM instead of read/write cycles to scratch files, then the speed would be drastically increased relative to the old version. Furthermore, there was an accompanying hypothesis that the implementation of other solving methods would further improve the speed of the code.

Professor Takahashi's penultimate working *VORLAX* compile was from 2014 and maintained reliance on the undesirable scratch files that existed as a legacy means of temporary memory storage. For historical context, on the machines *VORLAX* was originally designed for, random-access memory (RAM) did not exist in the way we know now, where every machine has in excess of 8gb available for no apparent reason. The memory was both expensive and physically large, thereby making it a very limited commodity. Thus, programs such as *VORLAX* utilized tape drive systems for the storage and manipulation of vectors. *VORLAX*, in particular, made use of six tape drives for arrays in its operation. Not all were used simultaneously, but in certain cases it was possible that all six could be used. When data was written or read to or from these tapes, the tape would physically rotate as the computer pulled the data,

and as such the array size limitations were limited by the physical tape dimensions. The fact that the tape drives had to physically rotate did restrict the speed of the drive, and so even though the program was very efficient, its execution certainly was not, especially by modern standards.

When *VORLAX* was moved from punch cards to more modern systems, the functionality relying on tape drives was retained. Fortunately, Windows has means for handling this by creating small files in the working directory that contain the data in a type of "simulation" (a term used loosely) of these drives. This was significantly faster than the old tape drives because modern hard drives will rotate from 5,000-10,000 RPM, and modern solid-state drives forgo any moving parts. However, the interfacing to these components is still slow compared to the interfacing to RAM. Thus, the proper course of action was to move *VORLAX* to an in-memory mode of operation.

As mentioned before, there was an outstanding goal of maintaining compatibility with this code. Professor Takahashi alone has dozens of programs that can work with *VORLAX* in different ways, and it was not productive to force change for the sake of change. Thus, it was decided that mimicking the operation of the tape drives via indices was an effective method of improving the code. *VORLAX* already had numerous COMMON blocks, used to store global variables for use between its numerous subroutines. Thus, two new COMMON blocks were created, each representing three of the antiquated tapes. Two blocks were used to stay within the memory limits imposed by the blocks. The COMMON blocks act as a vessel for moving the data between subroutines, but there are not cases where the code will be altering multiple tapes simultaneously. Thus, the operation was completed by referencing the COMMON blocks in each relevant subroutine. Because these blocks were representative of old tape drives, they were sized accordingly.

The sizing of the COMMON blocks was not a trivial task. *VORLAX* exists as a 32-bit program because leaving it as 32 bits is better for speed and backwards compatibility. This was deemed numerically permissible because *VORLAX* typically only reports figures to a maximum of five decimal places for floating-point numbers. However, one caveat of the 32-bit program with Windows is that there is an imposed 2gb memory limit on them. Thus, having arbitrarily large memory allocations was not possible (nor was it the best course of action). Upon inspection of the code, it became apparent that *VORLAX* was storing arrays of maximum size N×N in these tape decks. Thus, each COMMON block storage vector was sized as [1×25E+06], which allowed for a theoretical limit of 5000 array elements. This number of elements far exceeds a practical *VORLAX* model, and it also allows the six arrays to be allocated within the 2gb limit necessary for operation on Windows systems.

The utilization of these storage arrays behaves nearly identically to a tape deck system. For each subroutine that relies on these, there is a local counter variable that tracks the location of the read/write vectors in a way that is analogous to a tape array physically rotating. This was easy to implement as each read of a vector of length M would simply progress this counter the same amount. Likewise, for each "REWIND" command, the counter was reset to its starting index. This behavior was both efficient and avoided the introduction of any additional bugs or errors into the program.

Some may fairly question the choice of keeping things almost the same. The change to *VORLAX* made the program drastically faster (to be seen in a later section), which was already considered a massive benefit from the change. The reasons for keeping things as similar as possible trace back to the nature of FORTRAN77 and sections of code originating as FORTRAN IV [1]. The *VORLAX* program relies extensively on the logic behind GO TO statements. In modern programming, these are considered poor

practice, with FOR- and WHILE-loops being the preferred mechanisms for looping. Because of the existence of these GO TO statements, keeping track of the "location" of the virtual "tape" was desirable as it was a surefire way of maintaining the same operation and methods as originally assumed by the developers. Throughout the work with *VORLAX 2020*, there were many conscious decisions to exclude certain elements that had the potential to break the program, and this was one where it was deemed advantageous to err on the side of robust utility instead of "vast change". There may be future work in further condensing these arrays to work entirely in-memory, however much of the code operates via single dimension arrays that inherently rely on reading from a "tape" one vector length at a time. Thus, moving to 2-D arrays based only on individual variables was not considered wise for this time.

4.2 Solver Considerations

The accompanying aspect of the move to in-memory operation was the consideration of other solvers to improve the computation of the solution to the dense linear system. As mentioned in Chapter 2, there was an inherent challenge with the vortex-lattice linear system. Most modern solvers are not particularly fond of the dense system. As mentioned in the previous section, memory management was quite challenging, however it is inherent to the method. The system $\boldsymbol{A}\vec{x} = \vec{b}$ features a matrix A that is not sparse, positive-definite, symmetric, not diagonally-dominant, which severely limits the methods available, and for many of the methods that can theoretically work, practice shows the behavior to be erratic and prone to divergence. Heading into the studies between solvers, there was the understanding that convergence speed was second to reliability. Because *VORLAX* is used primarily in research, professional,

and educational environments, having a program that would randomly diverge for certain geometries was not an option. Thus, the following solvers were tested. The choice of solving techniques came largely from suggestions and anecdotal remarks from former attempts. Having restructured *VORLAX* to run quickly at this time, the following solvers were tested with the understanding that some were not explicitly suitable to this type of problem:

1. Controlled Successive Over-Relaxation
2. Purcell's Vector Method
3. Conjugate Gradients
4. Stabilized Bi-Conjugate Gradients Method
5. Intel Math Kernel Library LU Factorization and Solve

Of these methods, the Controlled Successive Over-Relaxation method and Purcell's vector method were the two originally available with *VORLAX*.

Each method was tested on the same setup on the same PC. This PC, a personal machine, features an Intel Core i9-9900k CPU (3.6 GHz Base Clock, 5.0 Boost Clock) with 32GB of DDR4 RAM with a clock speed of 2666MHz. *VORLAX* runs sequentially in the absence of any multithreading beyond any "black box" optimizations that the Intel FORTRAN compiler may implement during the compilation. When running the benchmark cases on the PC, the tests were completed under a "reasonable use" presumption. Thus, the PC was being utilized in the way a real-world engineer may do so, including having music playing, an email client open, MATLAB running idle, and a few Microsoft Excel worksheets. These overhead considerations fall far below what the PC is capable of running and almost certainly had little to no effect on the execution time of the program, though the considerations were made in order to mimic realistic usage cases.

Table 2: List of Grid Densities [9]

NVOR	RNCV	Total Points
25	10	250
50	20	1000
100	40	4000

In order to run the benchmarks, a minor alteration was made to the *VORLAX* program. Typically, the program is run via a command window spawned via Visual Basic with Microsoft Excel. The program is generally delayed when calling from Visual Basic (an initialization delay that does not exist when a user directly calls the program from the command line manually). This automation does lead to a short delay when calling the program, however, the interest of these benchmarks was to find the most efficient solver, and thus it was appropriate to add two lines of code to time the program execution. The timer begins before reading the input file, and it terminates right before the end of the program. The resultant timing value is printed to the same log file as the other data, allowing the user to record the runtime more accurately than an equivalent timer run in the Visual Basic script. This alteration was deemed inconsequential to the overall execution time of the program, as it is not called upon in any loops.

It should be noted that this delay indicates that, should a user wish to analyze multiple configurations, it is significantly faster to do so in a single *VORLAX* input, where the different configurations are handled internally. In a case where a user makes the decision to run each Mach number independently, for example, there would be significant time wasted both in the calls to the program, but as well as time wasted on redundant calls to the subroutines that parse the input file and set up the geometric relations for the solver.

The benchmark test case was a single AR = 20 infinitesimal flat-panel wing run in three different grid configurations. Because the focus was on the runtime of the solver itself, this was the only configuration tested, as it focused primarily on the N×N nature of the linear algebra problem. The three grid densities are listed in Table 2, where each subsequent grid featured four times the number of control points as the previous. The model was run at three freestream Mach numbers and three angles of attack, giving nine total configurations. The choice of multiple Mach numbers and angles of attack was deliberate and has to do with the internal structure of the *VORLAX* code. The program utilizes nested FOR-loops, where the external loop is through the Mach numbers and the internal loop is through the angles of attack. The reason for multiple values for each was to ensure that the overall effectiveness of the solving method was evaluated. For instance, if method "A" has a large overhead when allocating memory, then it may perform disproportionately well for a very small problem. Conversely, if method "B" features a small overhead when allocating memory, it may take longer to initialize but make up for the delay with faster solutions.

4.2.1 Controlled Successive Over-Relaxation

The CSOR method is a variant of the Gauss-Seidel iterative method for solving linear systems, and by extension is a variant of the Standard Over-relaxation method commonly used in computational fluids. The implementation in *VORLAX* allows the user to define the preferred relaxation parameters in the input configuration, defaulting to 0.10 if no value is specified. Generally, 0.10 to 0.20 have been shown to work fine in most applications of the code. The idea behind an SOR method is that, by defining a type of "relaxation parameter", the code may converge faster than

a standard Gauss method. Unfortunately, these methods are prone to convergence overshooting, which may lead to instabilities.

By utilizing a CSOR method, there exist two relaxation parameters, one which will "over-relax" the iterations, allowing convergence to accelerate, and one which will "under-relax" the iterations, serving as a type of dampening for the solution. These parameters are given via

$$\omega \equiv 1 \pm REXPAR \tag{4.1}$$

VORLAX computes the appropriate value automatically within the solving routine. For each iteration, the code compares the new residual magnitude to that of the former iteration to decide (based on the magnitude of the current solution "guess") whether to over- or under-relax the subsequent iteration.

The convergence criterion is based on a small percentage, approximately 0.5% of the root-mean squared (RMS) value of all of the values of all of the horseshoe vortex circulation strengths. By using this method, there is assurance that the magnitude of converge error (ϵ) does not become too large, while also ensuring that its magnitude is sufficiently small as to confirm proper convergence. The method of taking 0.5% of the RMS value appears "ad hoc" in the original implementation, but further inspection shows that the method is reasonable, and it exists primarily to ensure that the value is sufficiently small to satisfy convergence without affecting the robust nature of the *VORLAX* program.

The Controlled Successive Over-Relaxation (CSOR) method was the most popular option within *VORLAX* prior to the memory enhancements. This method was already reasonably quick, even before the changes. As a benchmark to the new capabilities, the original CSOR was run with only one change to the method. In the previous version of VOLRAX, the method implemented an initial guess for each iteration that

Table 3: CSOR Timing [9]

	NVOR = 25 RNCV = 10	NVOR = 50 RNCV = 20	NVOR = 100 RNCV = 40
VORLAX 2014	0.078130	0.984380	20.296880
VORLAX 2020	0.046880	0.328130	5.718750
Percent Improvement:	**40.0%**	**66.7%**	**71.8%**

was simply the zero-vector of the size of x. This was improved by changing the code to utilize the previous geometry configuration's answer as its initial guess (recall that the code allows for up to 16 Mach numbers with 16 angles of attack). What was a minor change did amount to a large improvement on subsequent calls to the solving routine.

As the revitalized *VORLAX* program saw more usage, and specifically more testing as to which aspects of the code were most time consuming, it became clear that using the previous answer was not always a good initial guess. For the angle of attack changes, the guess works great, as the program is dealing with a slight perturbation instead of a complete reconstruction of the linear system. However, when changing Mach numbers, the initial guess led to an order of magnitude increase in the solver time. Thus, it was beneficial to alter this functionality, now having the program reset the initial guess for each Mach number to a zero vector with only a single "1" element in the first slot. This greatly improved the runtime of the code, with a 33% decrease in program execution time for a medium grid at three angles of attack and three Mach numbers.

Without other significant changes to the operation of this subroutine, there were large performance gains because of the change to the in-memory operation. Table 3 shows the performance differences between the old version (*VORLAX 2014*) and the new version (*VORLAX 2020*).

The new version of *VORLAX* featured runtimes which were 40% faster at minimum, with a 71.8% improvement for the densest test configuration. As the grid sizing, N, becomes smaller, there is a limit to the time savings via the new in-memory operation. This is due to the fact that the routines necessary to generate the geometry take a non-negligible amount of the "wall time" on each call of the program. Thus, there exists a point where even if the solver was instantaneous, there would be no improvement due to the overhead of the preprocessing subroutines. This theory is further supported by the data from the dense grid sizes, in which the performance gains become much more apparent.

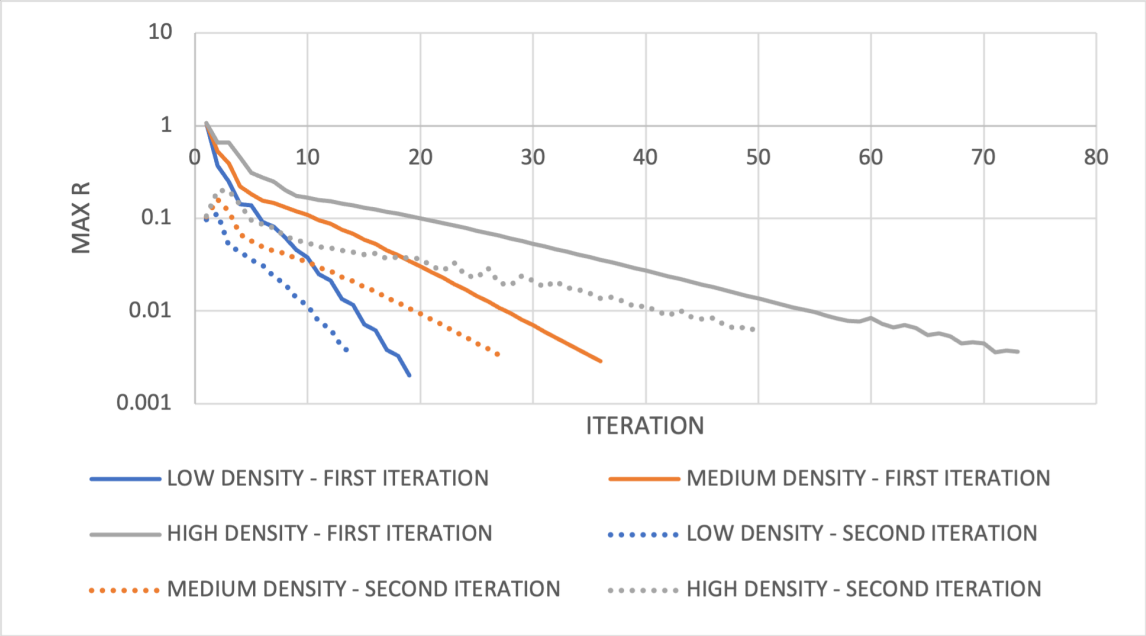


Figure 6: CSOR Residual Behavior [9]

Looking at the convergence plot of the CSOR method, there are two key details. The first (obvious) detail is that the rate of convergence is a function of the problem size. Thus, as the grids grow, the method will take longer to solve, as is expected. There is also the fact that when the second configuration is run, the implementation of

the improved initial guess as equivalent to the previous answer has a significant impact on the convergence speed. While the order of convergence (as seen in the semi-log plot) remains the same, the initial residual is $\mathcal{O}(10)$ better than the first call to the solver. This difference amounts to savings of approximately 25% fewer iterations, which amounts to a significant difference for a large number of configurations.

Fortunately, the CSOR method is incredibly fast. There are understandable perks to using this method. For instance, because the method was originally developed by Miranda [1], the method is already constructed to make use of the 1-D array nature of everything else in the program. The method relies on numerous read/write cycles to the tape deck storage, though it has already been discussed how much faster the new storage format is. The method is as efficient as possible with memory, having to reliance on large 2-D arrays, and already working with the existing COMMON block arrays. Thus, the implementation of this method is painless, requiring no extra libraries and allowing the program to stay small.

4.2.2 Purcell's Vector Method

Table 4: Timing of Vector Method [9]

	NVOR = 25 RNCV = 10	NVOR = 50 RNCV = 20	NVOR = 100 RNCV = 40
VORLAX 2014	1.390630	40.218750	1370.531000
VORLAX 2020	0.156250	29.046880	N/A
Percent Improvement:	**88.8%**	**27.8%**	**N/A**

Purcell's Vector Method was the alternate solver originally included with *VORLAX* [1][18]. The method, while capable of providing answers, was quite slow to the point of being unusable. Unlike the above CSOR method, the Purcell Vector Method uses

an orthogonalization procedure to provide the user with a direct solution to the linear equations. In theory, this sounds like a great tradeoff – sacrificing speed for a more exact answer. In practice, there is not such a drastic increase that the slow runtime is justified.

Because of the implementation of this method, certain array sizes were necessary to compute the solution. Unfortunately, these array limitations made it impossible to test the largest of the *VORLAX* benchmark cases on the new version. Fortunately, it became abundantly clear that this method was not worth utilizing. In the smallest test case, the solver still took three times as long as the CSOR method to arrive at a solution that was not significantly improved from that returned via the iterative methods. The medium-density grid case was no different, taking $\mathcal{O}(100)$ times as long to compute the solution. Thus, it was deemed impractical, even without attempting the largest test case.

One severe drawback to the Purcell method was its reliance on two auxiliary arrays for its operation. This was detrimental as the program was already working within memory constraints and as such had little overhead to spare. *VORLAX* utilizes several arrays for multiple purposes, and as such removing others was not an option.

There is a chance that the Purcell method served a genuine purpose at some point in history, though the reasoning is not entirely clear. The subroutine features calls via read and write operations to two auxiliary tape drives. Thus, there is a chance that for small problems, there may have been applications where the method was appropriate. When running on modern systems, there is zero benefit to using this method, and even if the user desires a "perfect" direct solve, there are better options, such as those offered by LAPACK.

4.2.3 Conjugate Gradients Method

The conjugate gradients method (CG) was one of three Krylov-based techniques. These methods have been heavily documented by Henk van der Vorst, whose publications were used extensively in these experiments because of their thorough nature and detailed algorithms for each method [19]. This method operates as a type of "gradient descent", where the information of the residual value is used to make an estimate of the solution vector. Much like the CSOR method, this method is iterative, which was deemed computationally advantageous after witnessing the Purcell Method.

There are some large caveats to the CG method. The method assumes but does not explicitly require that the matrix be sparse, for which the *VORLAX* matrices are not. There is the requirement that the matrix is positive-definite, which the *VORLAX* matrices are not. Finally, there is the assumption that the matrices are symmetric, again something the system in *VORLAX* is not. With these caveats, one may ask why the method was even attempted, and the answer boils down to anecdotal evidence, where a former user of the program spoke of a CG implementation. Because of this and the fact that the effort to develop the CG implementation was trivial, it was attempted.

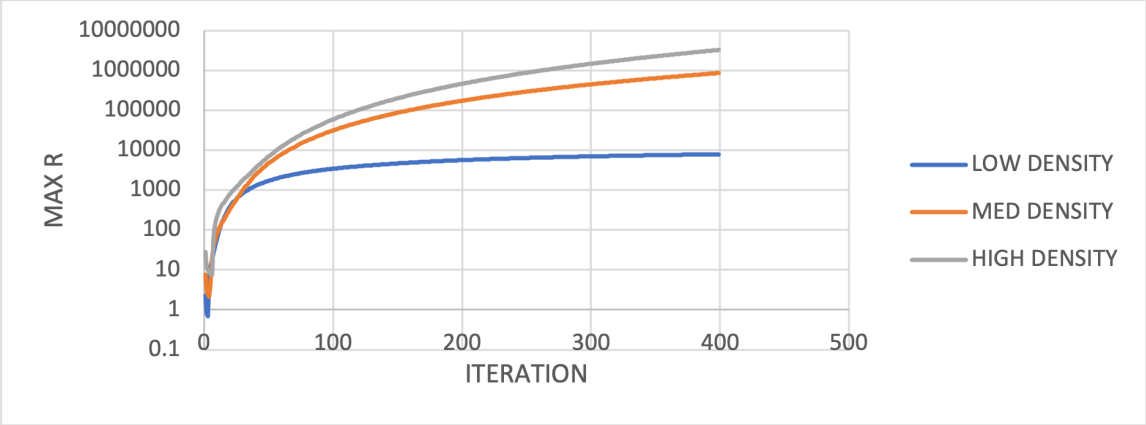


Figure 7: Conjugate Gradients Residual Behavior [9]

As expected, the method was unreliable and not able to be included as a solver in the program. The residuals plot shows the exponentially divergent nature of the method. Because of this divergent behavior, it was clear that the method was not suitable for *VORLAX*. Furthermore, because of the iterative nature of the method, the process would iterate until the user-defined iteration, making it a very slow method. Thus, little was expected from this method because of its slow and unreliable nature, however, it served as a great building block for the other Krylov-subspace methods.

4.2.4 Stabilized Bi-Conjugate Gradients Method

Due to the failure of the CG method, a more advanced technique was attempted under the guise that it would provide greater stability and be more applicable to the vortex-lattice problem. The Stabilized Bi-Conjugate Gradients (Bi-CGStab) method was supposed to do just this. In practice, Bi-CGStab is supposed to arrive to a final solution faster and smoother than a CG method [19]. This made the method attractive because of the unfriendly nature of the *VORLAX* system of equations. Bi-CGStab

Table 5: Timing of Bi-CGStab [9]

	NVOR = 25 RNCV = 10	NVOR = 50 RNCV = 20	NVOR = 100 RNCV = 40
Runtime w/o Preconditioning (s)	0.32813	4.03125	66.1875
Runtime w/ Jacobi Preconditioner (s)	0.625	5.64063	49.53125

removes the conditions that the matrix be symmetric, and thus is more applicable for a *VORLAX* implementation.

The method was implemented using Van der Vorst's algorithm [19]. Much like in the CSOR solver, the initial guess was chosen as the previous iteration's solution, unless it was the first run, in which case it was the zero vector with a single nonzero element to improve convergence speed slightly. Through testing with the benchmarking case, the method was found to be dependable, which was a massive improvement over the conjugate gradients method.

To further accelerate the solution, there was an attempt to implement preconditioning. However, this came with the restriction of keeping the code simple and not introducing outside libraries. Thus, the Jacobi preconditioner was chosen as a worthy preconditioner for the system due to its relative simplicity when computing the matrix inverse.

With the two variants programmed, tests were run to compare the results, giving interesting results. As a first remark, the method was quite slow, especially in comparison to the CSOR method. At its fastest, the method was nearly an order of magnitude slower than the CSOR method. Thus, it was quickly deemed nonideal for use in the *VORLAX* program. There was an interesting trend with the preconditioner, however, that highlights the relative speed of the program. When the problem size was small, as in the first two grid arrangements, the preconditioner introduced

enough overhead that the computation time was actually increased relative to the non-preconditioned time. However, when the problem size grew as in the third grid, the preconditioner saved a massive amount of time. This discrepancy can be seen in the semi-log residual plot.

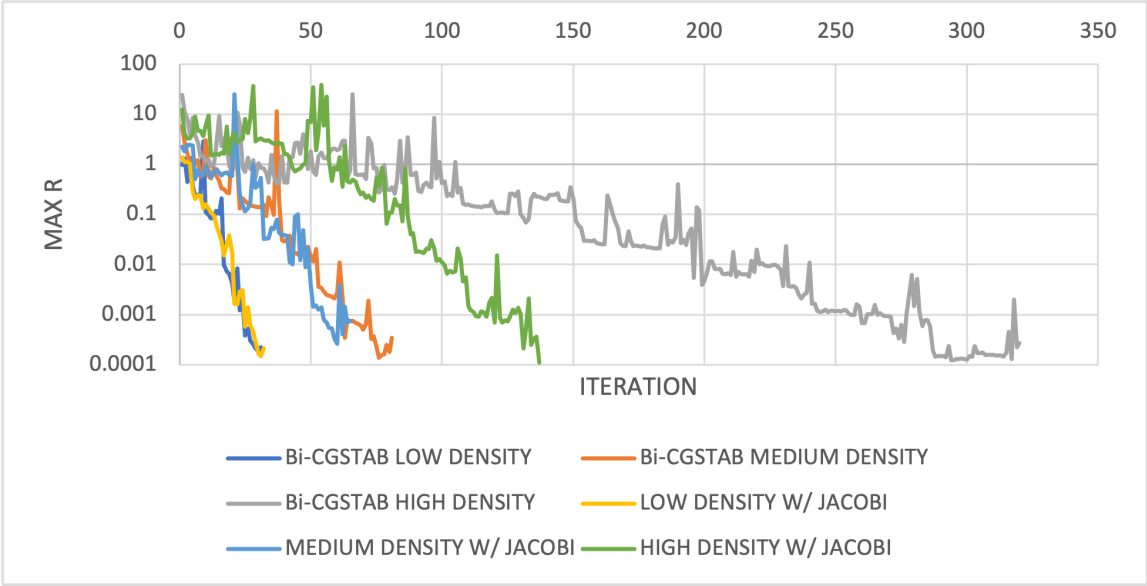


Figure 8: Convergence Behavior for Bi-CGStab [9]

In the two smaller grid cases, the convergence occurs at the exact same order both with and without the preconditioner. However, the large grid case shows a decrease of 183 iterations to hit its convergence. This indicates that as the problem size increases, the Jacobi preconditioner becomes more relatively valuable.

Another more concerning aspect of the Bi-CGStab method was that the convergence rate was much less smooth than that of the CSOR method. This impacted the choice of the convergence error limit, as the value was chosen to ensure that it was sufficiently low that a spike in the residual term would not trigger that. Nonetheless, even with

this lower limit, the Bi-CGStab method took longer to reach the same error magnitude as the CSOR method.

Both Krylov methods, the Bi-CGStab and CG methods, introduced unwanted complexity into the *VORLAX* program. Both methods were rather array heavy and therefore called for an additional COMMON block with several arrays necessary for their operation. This additional COMMON block was deemed preferable to having the large arrays initialized for each call of the solver function, as the latter option introduced compilation errors when dealing with the 32-bit memory management. Thus, the Krylov-subspace methods evaluated were not only slower, but they also introduced considerable complexity to the program, making them less than optimal for most cases.

4.2.5 Intel Math Kernel Library LU Factorization and Solve

While experimentation was being done with the iterative method, consideration was taken to test a direct solve method. Because this version of *VORLAX* was compiled working exclusively with the Intel FORTRAN compiler, an attempt was made to utilize the Intel Math Kernel Library (MKL). This library includes highly optimized linear algebra routines, providing a version of LAPACK that was optimized to run on Intel processors.

The MKL approach followed that of an introductory numerical methods course – there would be a call to a routine that would complete an LU factorization of the matrix followed by a subsequent call to a routine that would utilize this factorization to compute the solution. The hypothesis for this method was that, should the method

Table 6: MKL Timing [9]

	NVOR = 25 RNCV = 10	NVOR = 50 RNCV = 20	NVOR = 100 RNCV = 40
Intel MKL Wall Time (s)	0.04688	0.39063	8.95313

truly be optimized to the maximum extent possible, there may be an appreciable performance improvement by using the Intel solver.

Upon testing, the Intel Solver was surprisingly close to the CSOR method in runtime, showing that it was indeed well optimized for a direct solver. The Intel Solver, despite being a full LU-decomposition direct solver, performed nearly on-par with the CSOR method. This was unexpected, as oftentimes the direct solve will be slower than the iterative solve. However, on many of the cases the Intel solver was still slightly slower than the CSOR.

One undesirable caveat of the MKL solver was that it did introduce a new library into the *VORLAX* program. The addition of an extra library was deemed non-ideal; however, it would be worked around if the method proved to be exceptionally fast. The reason extra libraries were to be avoided with *VORLAX* was primarily to do with its usage case. Recall that *VORLAX* is a program meant for education, research, and design applications. It may be run on numerous different systems, some old, some new, all with differing computational power. Including the MKL solver introduced further complexity in the form of DLL requirements or a larger distributable size, neither of which were desired for the program. Ideally, *VORLAX* is a tool that can be run on a thumb drive with no external requirements. It is most practical as a single executable file, and thus further complicating its installation requirements was not worth it for a direct solve method that was not faster than the existing CSOR method.

Furthermore, the MKL solver still had requirements similar to those of the Krylov methods, introducing the need for more COMMON blocks and memory allocation that slowed the program at its initial launch. Furthermore, *VORLAX* is a tool that has existed for a long time without the usage of external libraries. Thus, there is an inherent disadvantage to introducing a new solver in the form of a closed source, "black box" solver. While it is likely that the MKL solver provides highly optimized, direct solutions, there is not a large difference in solution precision compared to the CSOR method, and there are not considerable time savings.

4.3 Solver Consensus

Thus, having tested numerous solving methods, both iterative and direct, it becomes clear that the original CSOR method as developed by Miranda is remarkably good. The method provides accurate results extremely quickly, and more importantly it does so without any significant additions to the code. Overall, the CSOR method featuring better initial guesses was the best performing method and was guaranteed to be compatible with previous geometries and results, as it was the same method.

4.4 Other Time-Consuming Subroutines

Upon further investigation of the source code, it became apparent that the solving routine was not necessarily the only subroutine taking up significant runtime during the calls to the *VORLAX* program. There exists a routine called "MATRX", whose purpose is to generate the grid of control points, complete with all of the normal and axial wash coefficients, as well as the influence coefficients for each vortex at

each control point. Thus, for sufficiently large problem sizes, the routine will need to calculate 5000 influence coefficients for each of 5000 control points, which is a large computational burden. This problem is inherent to the nature of the vortex-lattice method, without much possible to alleviate its cost.

To better understand the tradeoff of MATRX versus the solving routine, *VORLAX* was altered slightly to produce timings of each call to the two routines, and the results made it clear that the MATRX routine was more expensive per call than the solving subroutine. However, the solving routine was called for each Mach number and angle of attack combination, while MATRX was only called for each Mach number. Thus, sometimes the solver took up the majority of the time, while other times the MATRX routine took the majority of the time – dependent on the usage scenario.

There are three common usage cases for *VORLAX*. The first is using the program to model a configuration to run at a single angle of attack and a single Mach number – commonly seen in advanced wing design applications. The second is to model a body at a single Mach number and multiple angles of attack in order to generate lift and drag polar diagrams. The third common case is to run a small number of Mach numbers, usually about three, and multiple angles of attack, which is used to generate data regarding the stability and control of an airframe. Each of these configurations was tested with the modified "stopwatch" version of *VORLAX* in order to quantify the amount of time – both absolute and as a fraction of total runtime, that was spent running both the MATRX and solve routines.

These tests were completed using a Visual Basic script to automate the input file generation and run timings. Each Mach number and angle of attack configuration was tested for grid densities ranging from 250 total control points to 5000 total control points. Because this was primarily a test of the numerical methods – the time which

was necessary to construct the linear system and solve it, the exact spacing of the grid points was not of much concern. It should be noted that some of the plots show jagged trends for the small configurations. This is due to the CPU_TIME function used within the FORTRAN code to track the time. As the time to run became incredibly small, the accuracy of the timing function became inaccurate due to the sheer speed of program call – in some cases reporting 0.0000s for the calls. This is obviously incorrect; however it serves as a testament to just how fast the *VORLAX* program can be for simple cases.

The first tests were run for the single Mach and single angle of attack configuration. Without much surprise, the runtime grew exponentially overall as the number of grid points increased, seen in Figure 9. This was expected due to the nature of the vortex-lattice method, wherein each point is fundamentally dependent on each other point in the model.

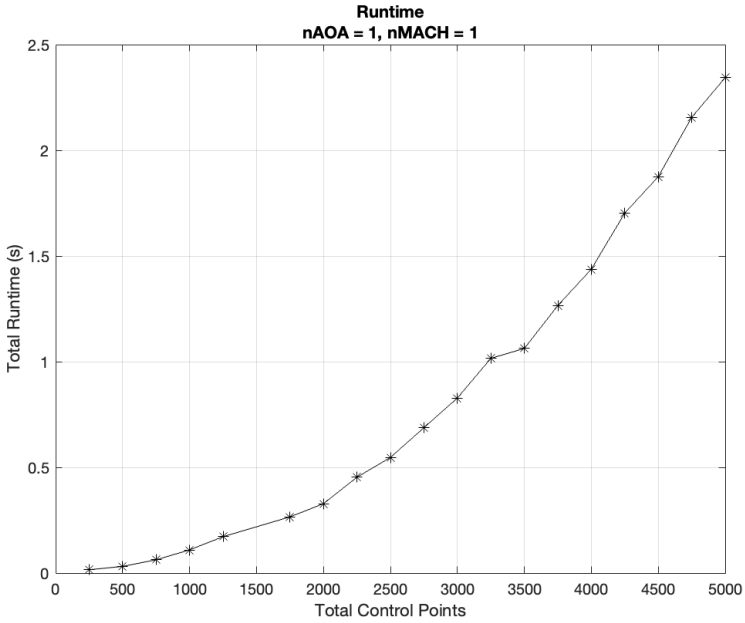


Figure 9: Runtime Growth with Problem Size for Single Angle of Attack, Single Mach Number

Having the theory that the MATRX and GAUSS subroutines were the most computationally hungry of the subroutines in the *VORLAX* program, it was desirable to quantify the amount of time per each as a fraction of the overall runtime.

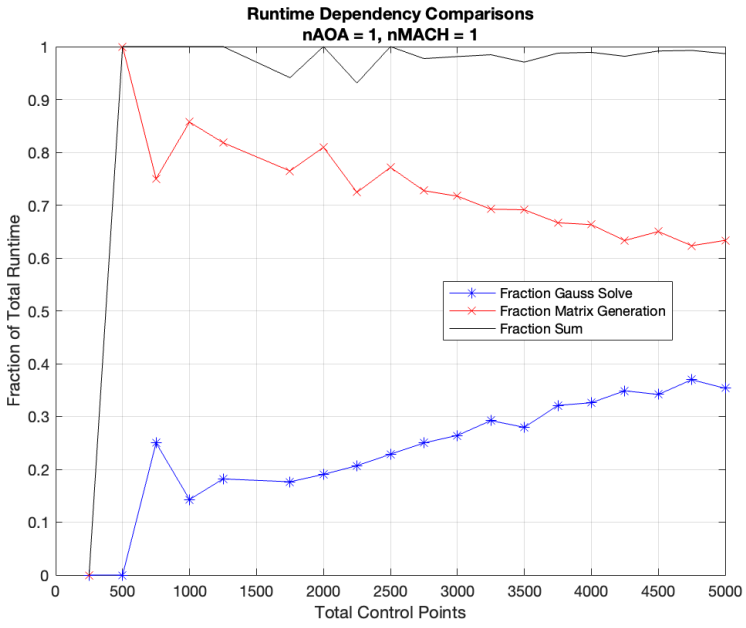


Figure 10: Comparison of Runtime Dependency for Single Angle of Attack, Single Mach Number

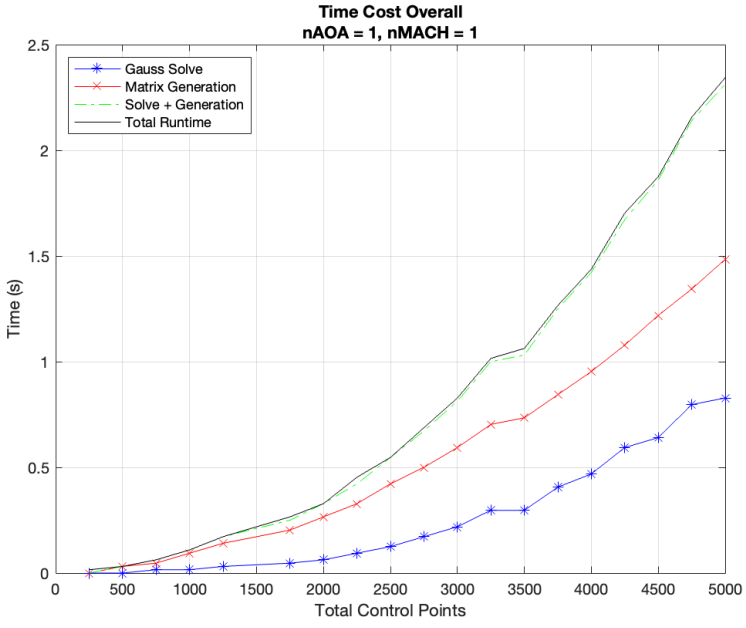


Figure 11: Overall Time Cost for Single Angle of Attack, Single Mach Number

Figure 10 is telling, as it becomes clear that the MATRX subroutine is not a negligible contributor to the overall runtime! Understanding that there are some minor errors in the timing because of the intrinsic timing function, the overall trends are still very obvious. For each of the grid densities, both the MATRX and GAUSS subroutines are overwhelmingly the driving force of the total runtime – accounting for over 90% of the time allocated to each program call. Figure 11 shows this proportion as absolute time, rather than via fractions. Thus, it is clear that the two most expensive calls account for almost the entirety of the runtime. While this run shows that the MATRX call is more expensive, the story changes when running more angles of attack relative to Mach numbers.

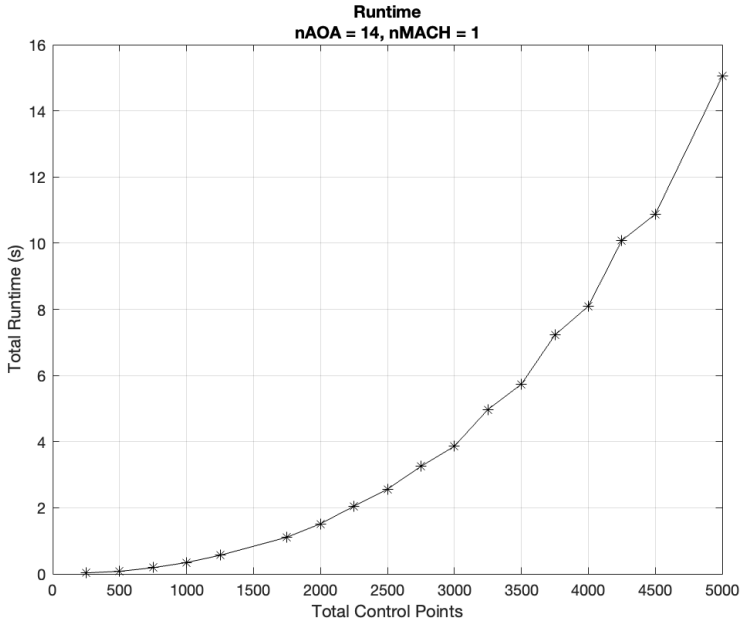


Figure 12: Runtime Growth with Problem Size for Fourteen Angles of Attack, Single Mach Number

The second usage case was to run a single Mach number at multiple angles of attack, in this case 14 total angles. Recall that while MATRX is called for each Mach number for most use cases, the GAUSS subroutine is called for each angle of attack for each Mach number. Figure 12 shows the evolution of the total runtime. As expected, the time still follows a parabolic trend based on the number of control points, and the run takes longer due to the additional pitching angles.

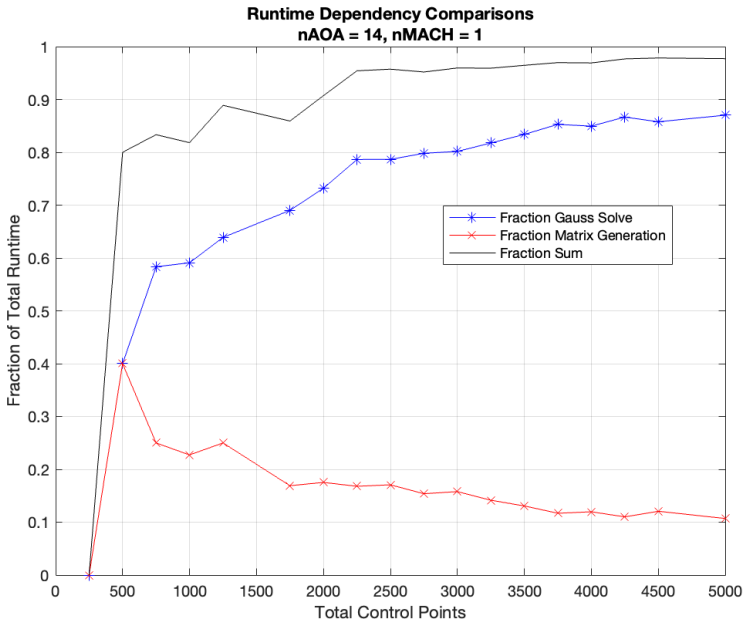


Figure 13: Comparison of Runtime Dependency for Fourteen Angles of Attack, Single Mach Number

Figure 13 shows the fractional breakdown of each portion of the code. It is noticeable that the MATRX and GAUSS subroutine calls still account for the majority of the runtime, however this majority is lesser. Recall in Figure **??** that the total fraction was north of 95% for essentially all of the control point configurations, but in this case it hovers slightly lower near the 80-95% range, indicating that the addition of pitching angles has led to increased overhead in other parts of the code.

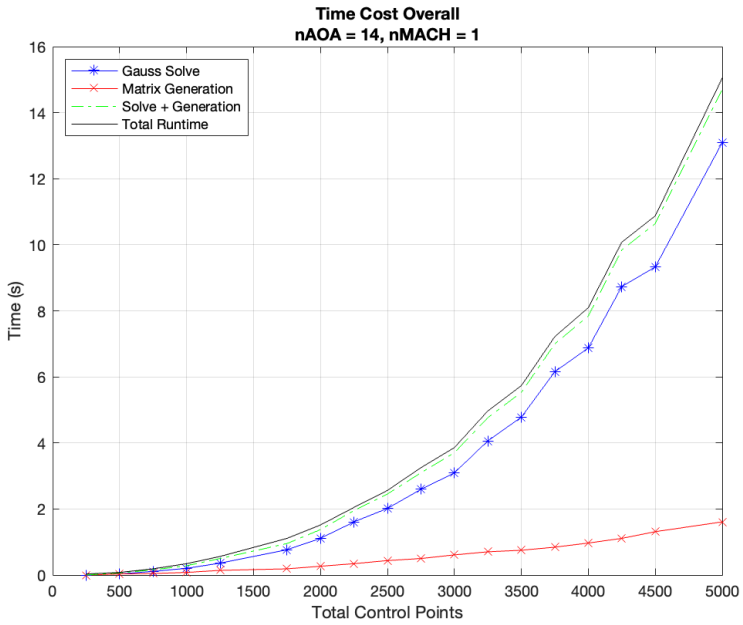


Figure 14: Overall Time Cost for Fourteen Angles of Attack, Single Mach Number

While the proportions may seem largely different, the bigger picture seen in Figure 14 shows that the two most expensive subroutines still account for most of the runtime. However, the bigger picture shows that it is now the linear system solver driving the total runtime instead of the system generation. The MATRX subroutine, while considerably smaller in overhead than the GAUSS subroutine, is not negligible, and can still benefit from improvements.

The final configuration was that commonly seen in stability and control applications, where the user will define a system with 3 freestream Mach numbers and 14 angles of attack. The array of pitching angles allows the user to compute the stability derivatives and having multiple Mach numbers will allow the user to interpolate between them to approximate performance derivatives in multiple flight conditions, such as takeoff, climb, cruise, descent, and landing. While this may seem like a lot, it is worth remembering that initializing the command window to run the *VORLAX* program on Windows can often amount to fairly considerable time contributions, thus it is advantageous to configure all of the desired combinations in a single file.

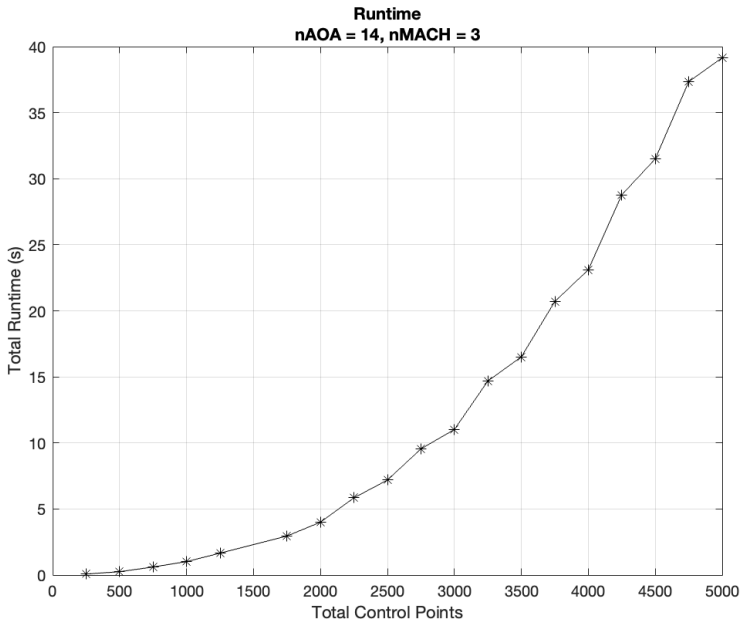


Figure 15: Runtime Growth with Problem Size for Fourteen Angles of Attack, Three Mach Numbers

Figure 15 shows the overall runtime of this test, once again demonstrating a parabolic growth in runtime as the number of control points increases linearly. This run took nearly 40 seconds to compute, and while this is remarkably fast by modern CFD standards, it begins to add up depending on the number of points. Commonly, the stability and control applications of *VORLAX* will involve the program running five different files, one of which is in a neutral configuration, one at a sideslip angle, and three with control surface deflections. Thus, for a "common" application, this may amount to a considerable amount of time. Previously, each of the stability and control files with 2,000 control points would take O(120) seconds to run (depending on the case), meaning that the whole stability and control case would take 10 minutes for each attempt! Thus, every time a tail surface changed even slightly, or perhaps the aileron surface area increased incrementally, there would be nearly 10 minutes wasted to downtime waiting for *VORLAX* to run. This was largely the inspiration of this work, and when considering this portion on timing, it is clear how helpful these improvements are in real-world applications.

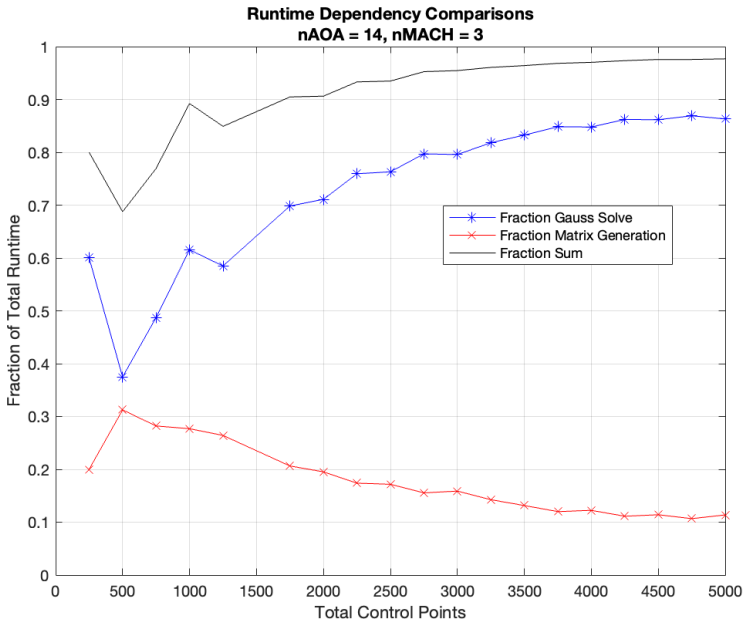


Figure 16: Comparison of Runtime Dependency for Fourteen Angles of Attack, Three Mach Numbers

Figure 16 shows the fractional contributions in the stability and control case. Once again, as the problem size increases, the solver and linear system generator become overwhelmingly dominant in the overall runtime. Interestingly, the proportions do not change much relative to the case with only Mach number, and this makes sense as the calls to the solver routine grow linearly as a function of Mach number, as do the calls to the matrix generation routine. Thus, it makes sense that the proportions will remain nearly identical, as they are linked inherently based on the structure of the code.

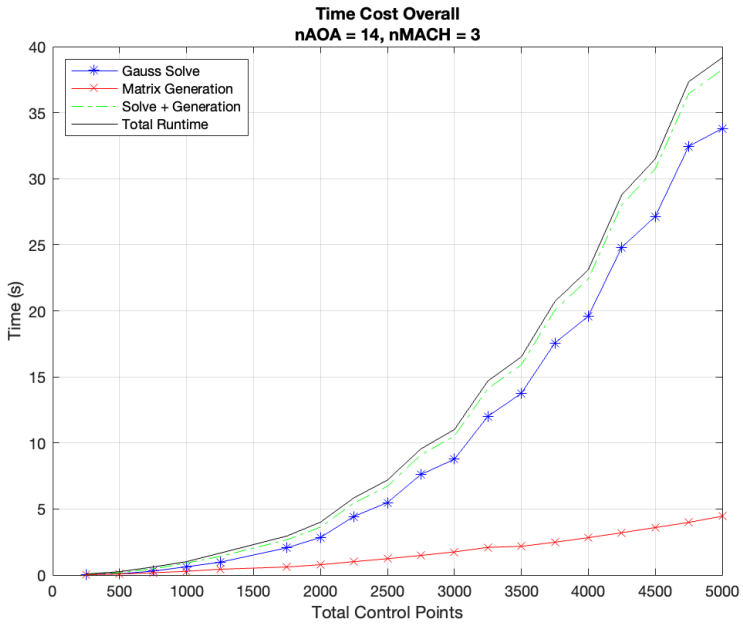


Figure 17: Overall Time Cost for Fourteen Angles of Attack, Three Mach Numbers

Figure 17 shows again that while the overall runtime of the program in this configuration has increased, the contributions from each subroutine remain more or less the same. Thus, for two of the common configurations, the amount of time spent solving the system is large relative to the amount of time constructing each system, except in the case of the single panel where the matrix construction was the more expensive operation. However, it is necessary to consider the overall runtime of the configurations tested. While the single panel shows a larger dependence on the MATRX subroutine, the total runtime is often significantly shorter. For instance, a very detailed wing design will often have 100 spanwise control points and 20 chordwise points, giving a total runtime on the order of 0.5 seconds, which leaves little to gain in terms of optimizations. Conversely, a complex stability and control model of a full aircraft may have nearly 4000-5000 control points (depending on the configuration and number of panels), which will take 30 seconds to run, but also

needs to be run five times, giving a total "wall time" of each case of 150 seconds. Thus, the potential to save time falls firmly in the solver subroutine, as even a 0.03% reduction in runtime would amount to more time than the single configuration takes altogether.

To complete this portion on numerical behavior of the *VORLAX* program, it is worth briefly reiterating some of the nuances that appear to be limiting the solving speed in its current state. There are some changes that may be experimented with to decrease the runtime, however each change risks altering the dependability of the code or changing certain notational norms carried from the FORTRAN 66 framework, and as such any modifications should be made with caution.

Some testing was completed within the timings which demonstrated a fair sensitivity to the initial guess of the CSOR solver. This indicates that there may be further improvements to be made by introducing a form of a multigrid solver, using coarse grid circulation solutions to accelerate dense grid solutions. This would take advantage of the exponential nature of the solver cost, potentially saving time overall. It may also be worth working with the array structures of the program to see if modern FORTRAN compilers can run more efficient on an array-based system. One hurdle with this is that the *VORLAX* program operates in a very vector-focused manner, and all of the arrays and loops are currently geared for single dimension iterators, so this would not be a trivial change. The program has a lot of life remaining, and Chapter 5 will show examples of just how useful the program outputs are.

CHAPTER 5

DETAILED PERFORMANCE AND FURTHER BENCHMARKING

The *VORLAX* program shows its strengths particularly in shock-free, compressible flows. Despite its long track record and usage history, validation tests were performed. By sheer nature of the program, there are hundreds of variables that one may choose to examine for the validation. This presents hurdles, however there are examples of empirical data that are possible to use to validate the updates to the *VORLAX* program. One such document is a NACA technical report with empirical data detailing macroscopic aerodynamic attributes of a swept wing [20]. The report features a half-span wind tunnel test at a variety of Mach numbers. This was used extensively to test the various operational modes of *VORLAX* and to understand the implications of various modifications to the input file.

The wing described in the technical report features inputs compatible with the *VORLAX* input structure, and as such it was possible to replicate the model almost exactly using the exact data in the paper [20]. This includes not only the general planform geometry, but the thickness and camber profiles as explicitly defined in the report. The comparison is shown in Figures 18 and 19.

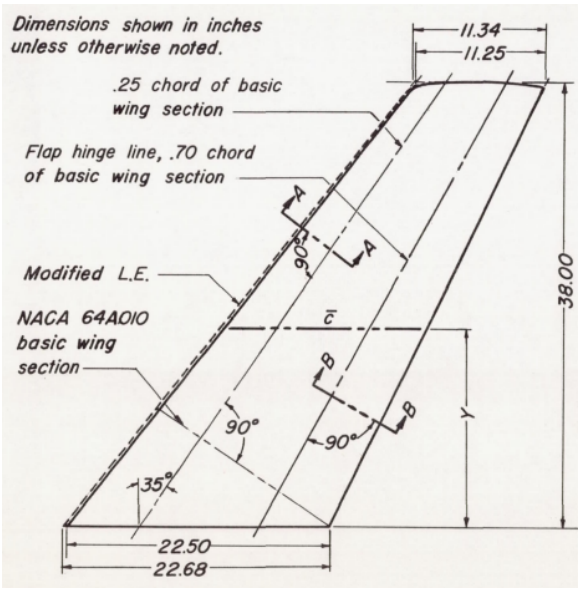


Figure 18: NACA Wing Dimensions [20]

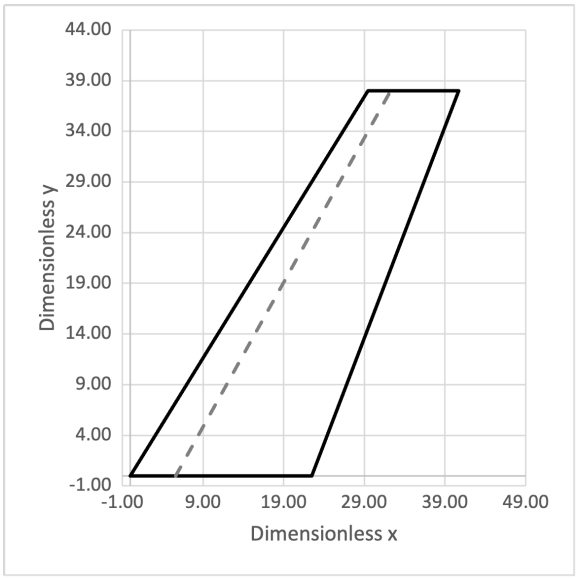


Figure 19: VORLAX Geometry for NACA Wing [9]

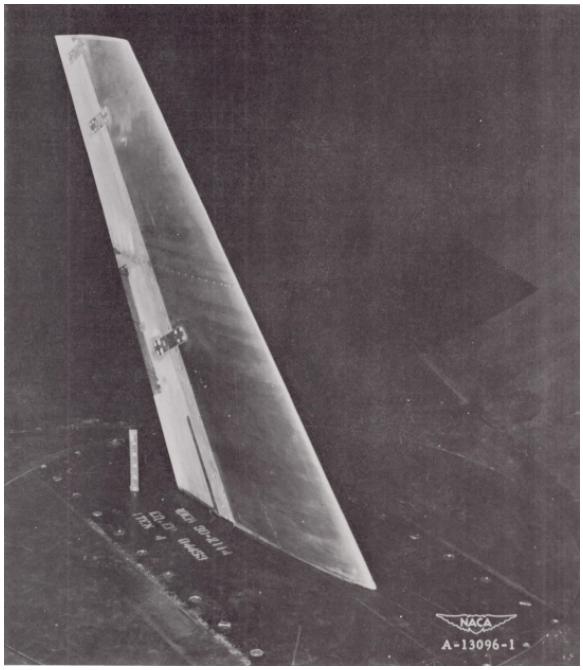


Figure 20: Image of NACA Wing in Wind Tunnel [20]

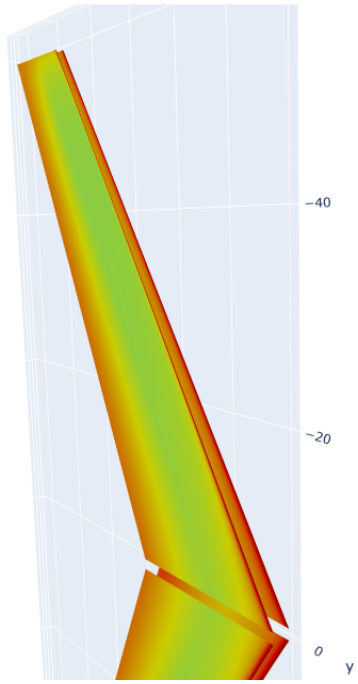


Figure 21: 3d VORLAX Drawing of NACA Wing

There were several parameters to verify in order to compare the data from the test to the wind tunnel results. Focusing on those primarily related to the program performance (i.e. numerical parameters instead of geometric alterations), varying the grid spacing, leading-edge suction, and spacing style in the spanwise and chordwise directions. These tests would be completed on a variety of geometries, including flat-plate models, cambered-plate models, and thick models of both nonplanar and planar varieties.

Other tests were completed to demonstrate the functionality of the *VORLAX* program for whole-airframe representation. These models are not basic, rather they include aerodynamic control elements such as ailerons, elevators, and rudders. As thoroughly covered, the possibilities of the VORLAX geometries are endless.

5.1 Grid Calibration

When running *VORLAX* in its "flat panel" configuration, the zero-mass flux condition must be resolved on each side of each panel. Recall that each panel is represented by cartesian leading-edge coordinates, and the user may specify many panels. For a common entire-aircraft representation, this amounts to 11 panels. Fortunately, it is trivial to automate the panel definitions to ensure that they align correctly and there are no gaps between them.

There are a given number of grid points specified by the user for each panel. VORLAX will space these points on the panel either linearly or via a cosine spacing that groups the points either equidistantly or with higher densities towards the leading edge and trailing edge. For practical subsonic applications where consideration of the leading edge thrust effects is necessary, the user must use the cosine spacing. Figure

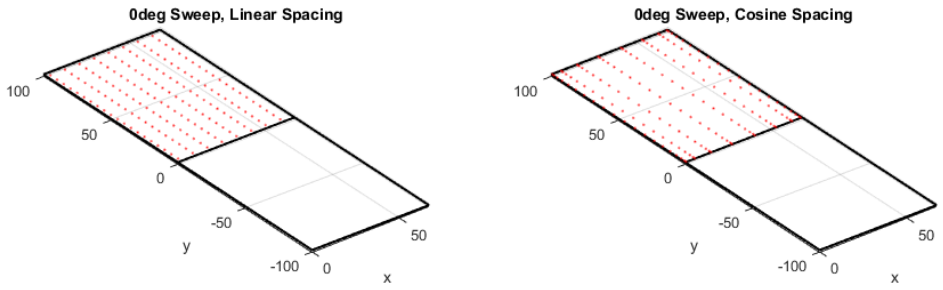


Figure 22a: VORLAX Grid Spacing with No Sweep [9]

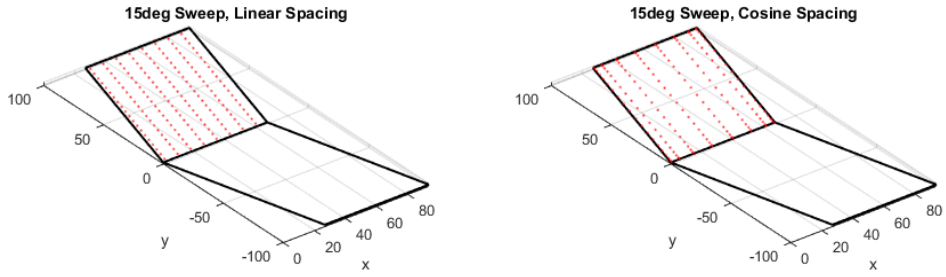


Figure 22b: VORLAX Grid Spacing with Sweep [9]

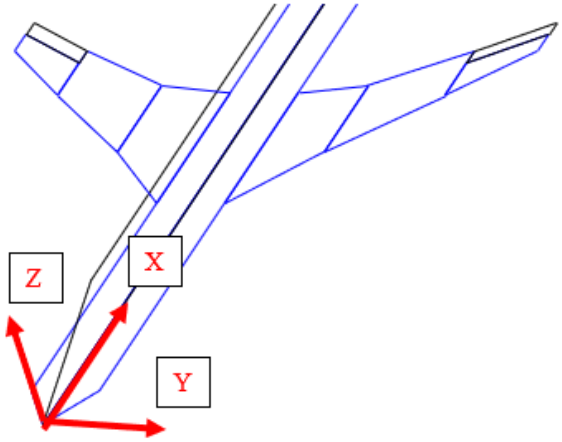


Figure 23: VORLAX Cartesian Coordinate Frame [9]

22 shows the differences between these spacing methods for the same number of points on both a swept and an unswept panel.

While *VORLAX* offers the option to use either linear or cosine spacing in the chordwise and spanwise directions, the methods are not created equal. As mentioned before, the leading-edge suction is calculated using Lan's method, as given in the theory guide [17]. However, when reading the documentation for Lan's method, it becomes clear that the method was developed for use with a cosine spacing in the chordwise direction, not a linear spacing. There were some cases where the linear spacing would be utilized under the assumption that the two spacing methods were more-or-less equal. This test shows otherwise, and actually demonstrates that the difference between the two methods is massive. Thus, to avoid future confusion and inaccurate results, the code was modified in order to reject the run if the user specifies a nonzero SPC value with linear chordwise spacing.

The flat panel configuration is remarkably accurate given its simplicity. It falls short when utilized for a proper wing design with critical pressure coefficient considerations, but otherwise provides perfectly usable data. For a basic first-order experiment to determine whether a wing design is able to work at all, the method will return results that may be easily carried on for usage with the more nuanced features of *VORLAX*. When testing the stability of a proposed aircraft design, the flat panel representation is adequate to provide the information about the control surface power to the point where the user can understand if their control surface sizing and placement is sufficient for the airframe.

The first example case of the flat panel method will be that of the example wing test case. While the Vortex-Lattice Method is relatively insensitive to the grid spacing, there are still grid densities that fall outside of the usable range. It was imperative to test and quantify these grid densities in order to determine a formally optimal grid spacing for this test configuration. This was completed by running the *VORLAX*

model at 20 different half-span grid densities, ranging from 5 to 100 grid points, while also testing chordwise densities ranging from 4 to 20 points, incrementing by two points. Each of these grids were tested at a single Mach number, M = 0.21, as well as a complete ranging of pitching angles ranging from -10 degrees to 16 degrees. This test was completed twice for each configuration, representing each combination of allowable spacing methods: both having cosine chordwise spacing, but one having linear spanwise spacing and the other having cosine spanwise spacing.

To determine the optimal grid density, *VORLAX* was run for the geometry of the test case for each of the 180 configurations. The test for optimal grid density was run for both a flat panel and for the three-dimensional panel, with the goal being to find an "ideal" spacing for the models. The tests needed to be automated, both to ensure consistency with the input files (as opposed to risking human errors on the inputs) and to run the tests quickly. The *VORLAX* trials were run using VBA, which then parsed the results into large text files that were convenient for plotting via MATLAB.

With each of the configurations loaded into MATLAB, the data points were compared to the results in the NACA report for the swept wing [20]. There were some small caveats with this approach, though they are minor. Using the *VORLAX* outputs, a curve was fit to the lift and drag polar charts, using the same curve fitting library as before – this was deemed inconsequential to the big picture of the comparisons. This was necessary to ensure that the ordinates aligned when attempting to quantify the agreement between the test and simulation curves. Additionally, the test data was never tabulated, and as such the measurements as seen by MATLAB came from careful tracing of the figures in the report. Finally, the initial NACA report utilizes only a half-span model in a wind tunnel using a wind tunnel correction factor, which is a third potentially significant source of error in the test data. While these may have

both introduced error, it is of a small magnitude that will be more apparent in the subsequent graphs showing the comparisons. In an attempt to obtain quantitative correlation metrics, the L2-Norm of the absolute error based on the curve fit was computed. From this, it was possible to obtain pessimistic information regarding how "well" a curve was correlated. However, it quickly became clear that the information computed by *VORLAX* was largely insensitive.

To match the drag coefficients, it was beneficial to add an approximation for the friction drag. In earlier attempts, the drag was incorrectly correlated using only the *VORLAX* results, though this showed only the induced drag figures, which are dominant at higher lift coefficients. For completeness, it was desirable to demonstrate the values of the *VORLAX* drag figures with minimal correction, which will be seen in the sections relating to single angle of attack results. The zero-lift drag was approximated using a simple form-factor approach, as specified in [8]. This method uses an empirical Reynolds number approach as a function of the mean chord length to approximate the friction drag. To do so, the standard atmosphere model given in [8] was used with the freestream Mach number, assuming the tests occurred at approximately sea level. While an approximation, this was perfectly acceptable for *VORLAX*, as it gave a benchmark that would adequately tell an engineer whether an idea was worth pursuing, all without adding considerable complexity beyond basic wing parameters.

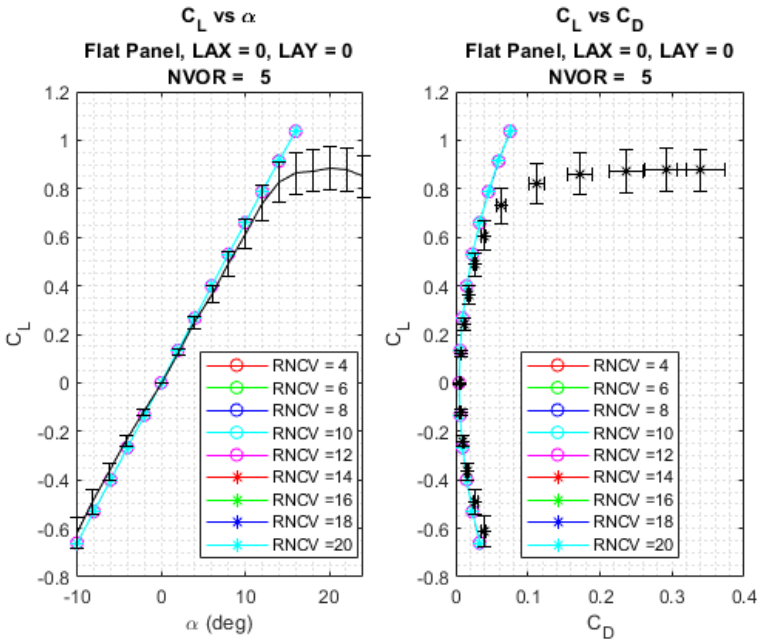


Figure 24a: NVOR and RNCV Comparisons for Flat Panel with Cosine Spacing, NVOR = 5

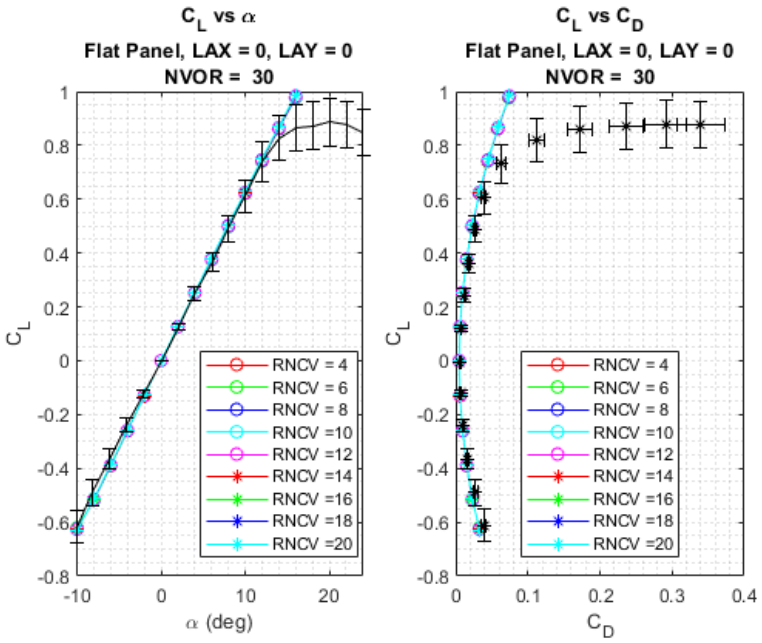


Figure 24b: NVOR and RNCV Comparisons for Flat Panel with Cosine Spacing, NVOR = 30

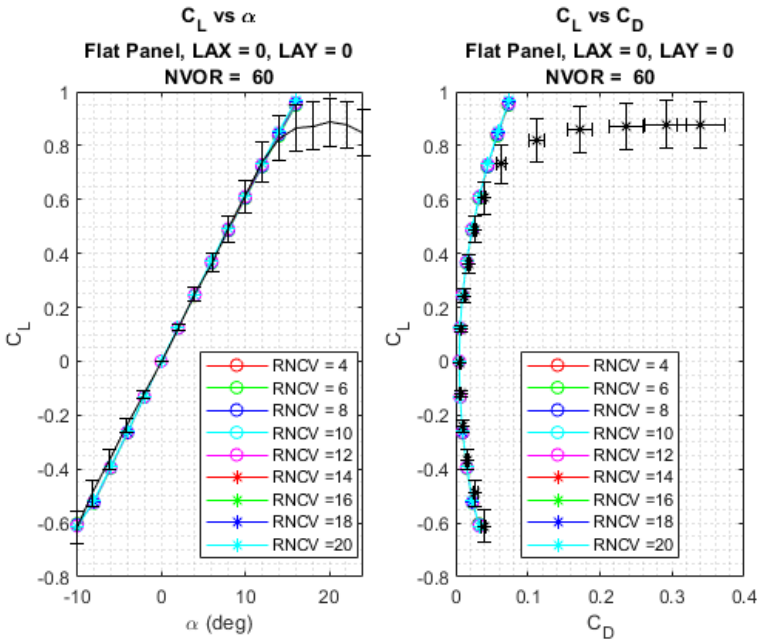


Figure 24c: NVOR and RNCV Comparisons for Flat Panel with Cosine Spacing, NVOR = 60

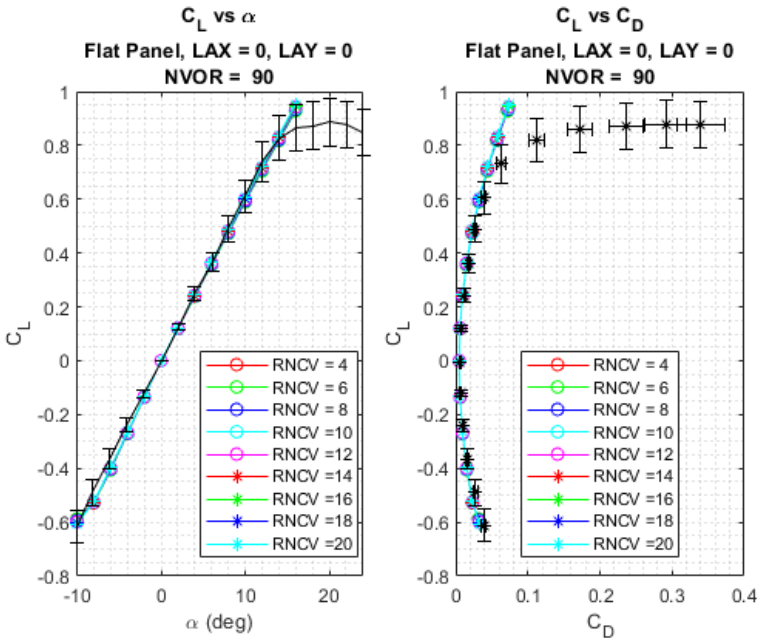


Figure 24d: NVOR and RNCV Comparisons for Flat Panel with Cosine Spacing, NVOR = 90

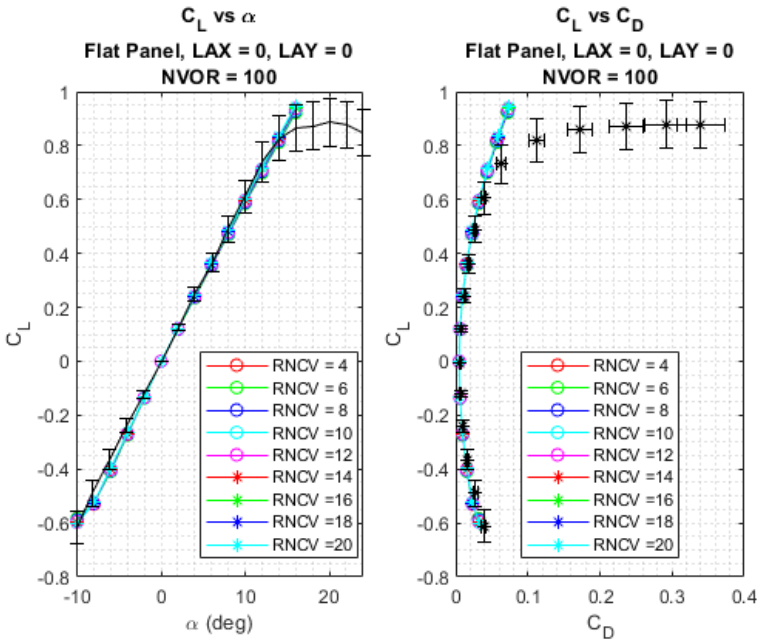


Figure 24e: NVOR and RNCV Comparisons for Flat Panel with Cosine Spacing, NVOR = 100

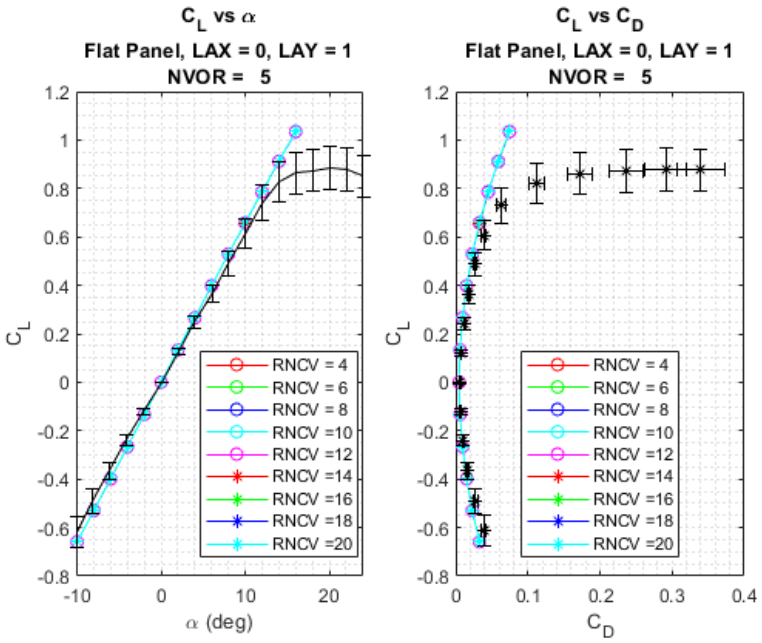


Figure 25a: NVOR and RNCV Comparisons for Flat Panel with Linear Spacing, NVOR = 5

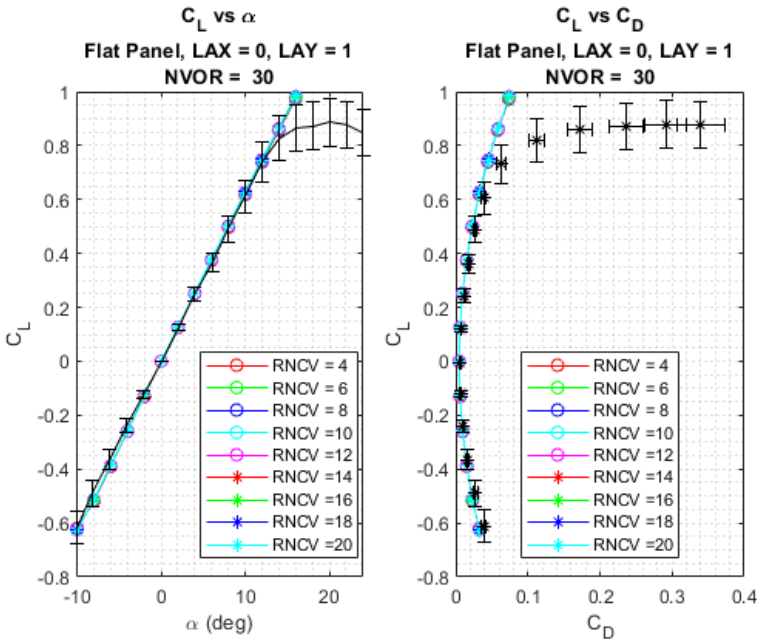


Figure 25b: NVOR and RNCV Comparisons for Flat Panel with Linear Spacing, NVOR = 30

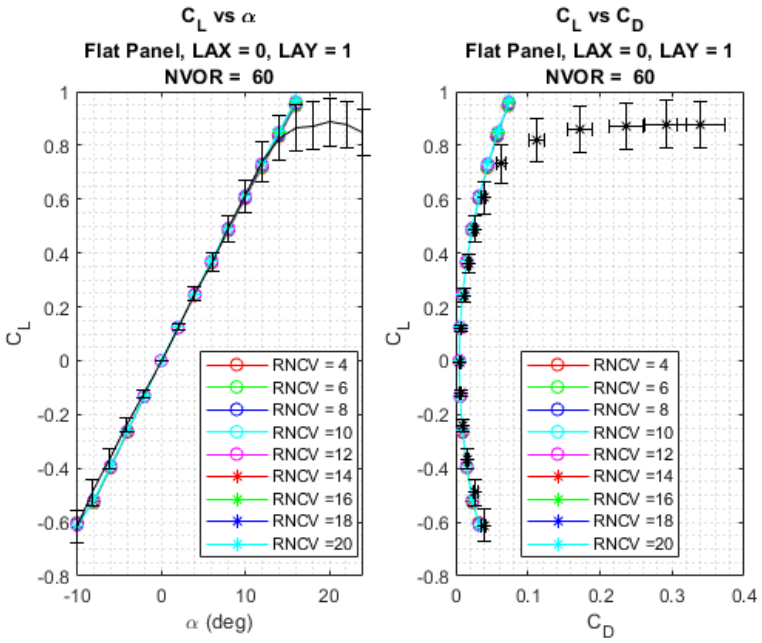


Figure 25c: NVOR and RNCV Comparisons for Flat Panel with Linear Spacing, NVOR = 60

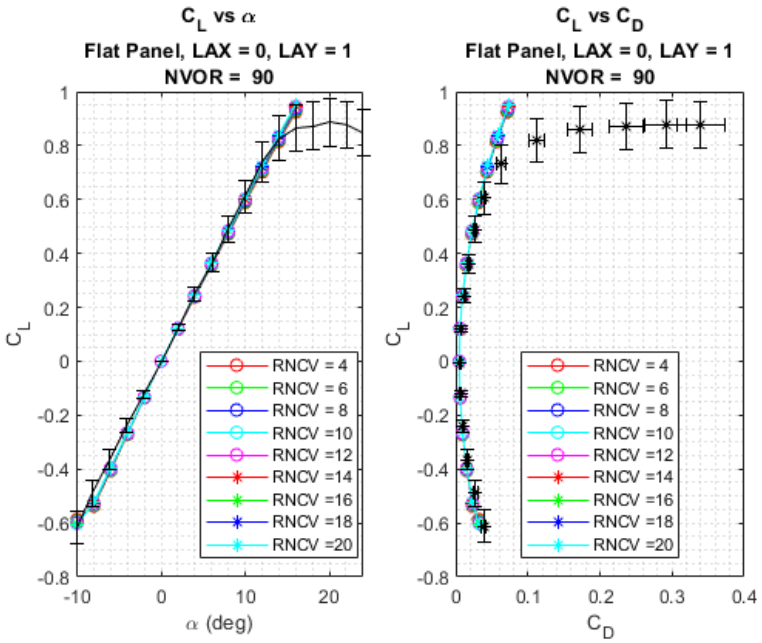


Figure 25d: NVOR and RNCV Comparisons for Flat Panel with Linear Spacing, NVOR = 90

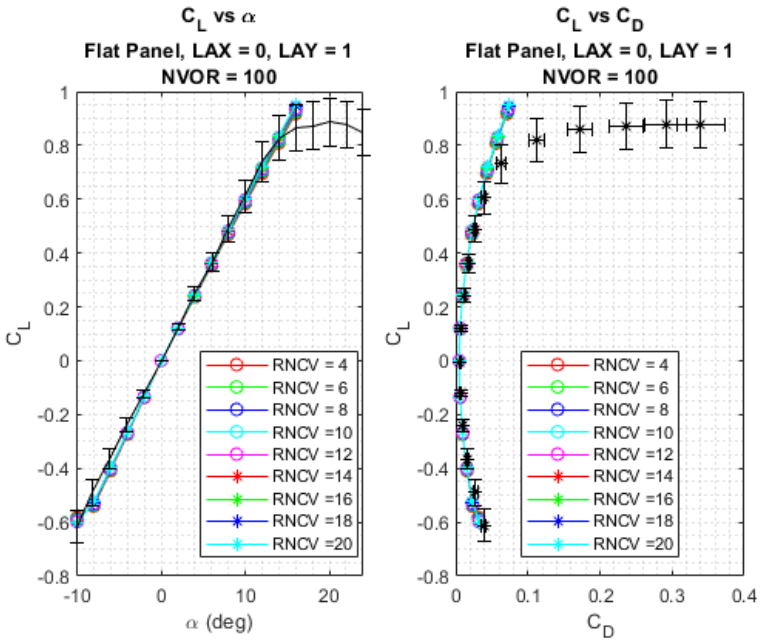


Figure 25e: NVOR and RNCV Comparisons for Flat Panel with Linear Spacing, NVOR = 100

Figures 24 and 25 show the results of the comparisons for both of the flat panel configurations – one which has cosine spacing both chordwise and spanwise, and another with cosine spacing in the chordwise direction and linear spacing in the spanwise direction. Beginning with the flat panel case with both directions dictated by cosine spacing, it is clear that the *VORLAX* output results are not affected by the grid densities. The trials were unbiased, so some of the attempts featured incredibly coarse grids or grids with very badly skewed grid aspect ratios. Nonetheless, the results all agree comparably – it is difficult upon visual inspection to point out a "best" model. When evaluating the L2-Norm magnitudes for each, the C_L plot shows a best grid of $NVOR = 60$ and $RNCV = 20$ with an L2-Norm of 0.0470, which the C_D plot shows a best grid of $NVOR = 100$ and $RNCV = 6$ with an L2-Norm of 0.0053. However, the magnitudes of the error are worth noting. While the CD plot has a spread of merely ± 0.0023 in its norm, the CL plot features a spread of 0.0739. So, when thinking practically, it is a safer assumption to work with a relatively small uncertainty in the drag measurements than to have substantial errors in the lift. Thus, in the interest of grid sensitivity, the best grid for the flat plate model is that with $NVOR = 60$ and $RNCV = 20$.

For the configuration with linear spanwise spacing, the results are nearly identical to those from the cosine spanwise spacing. Much like before, nearly all of the curves fall within the same margin of error, with no clear "best" grid spacing. The same correlation test was run in MATLAB with the same caveats as above, and the ideal grid was found to be $NVOR = 50$ and $RNCV = 20$ with an L2-Norm of 0.0462 from the C_L plots, and $NVOR = 100$ and $RNCV = 6$with an L2-Norm of 0.0054 in the C_D plots. Just like before, the values for C_D were massively less sensitive than those

for the lift coefficient, and as a result $NVOR = 50$ and $RNCV = 20$ were deemed the "best" configuration for the arrangement.

These same tests were completed using the sandwich panel configuration. However, this test presented more interesting results. The sandwich panels have two modes, one where *VORLAX* generates a truly nonplanar geometry with displacements in the $\hat{z}$-direction to the control points and bound vortices, and one where everything remains at a set z-value. The two modes are fundamentally different, and as such it was appropriate to re-run the tests for each. Fortunately, doing so would give values that would help with the latter portion of the numerical examples, as it is very redundant to generate pressure contours for each and every grid size.

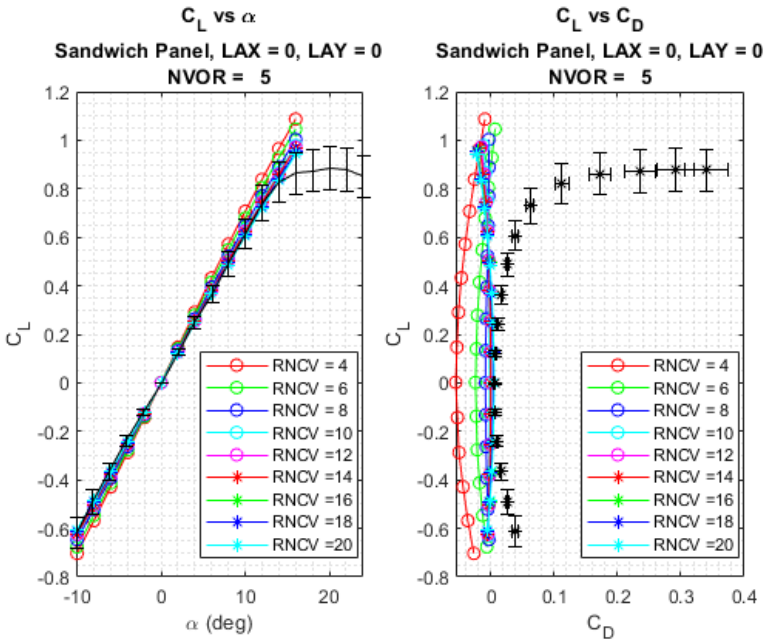


Figure 26a: NVOR and RNCV Comparisons for Sandwich Panel with Cosine Spacing, NVOR = 5

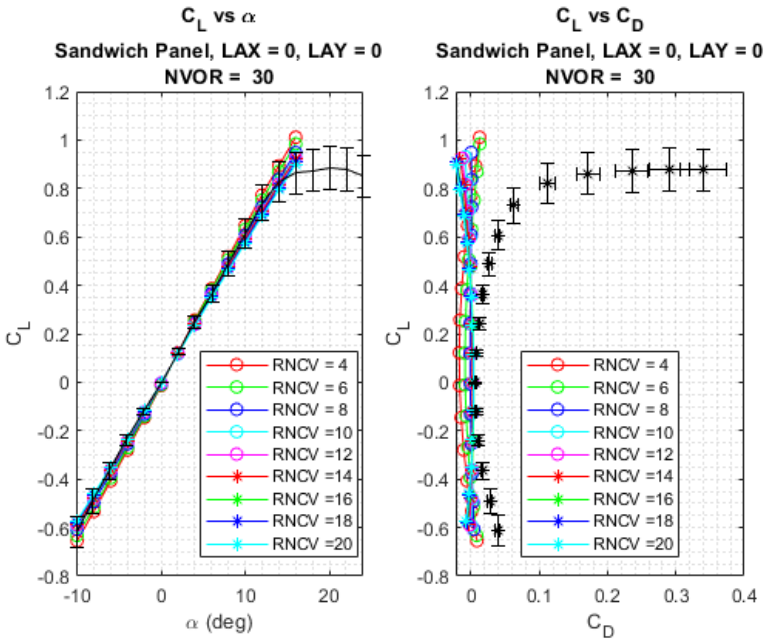


Figure 26b: NVOR and RNCV Comparisons for Sandwich Panel with Cosine Spacing, NVOR = 30

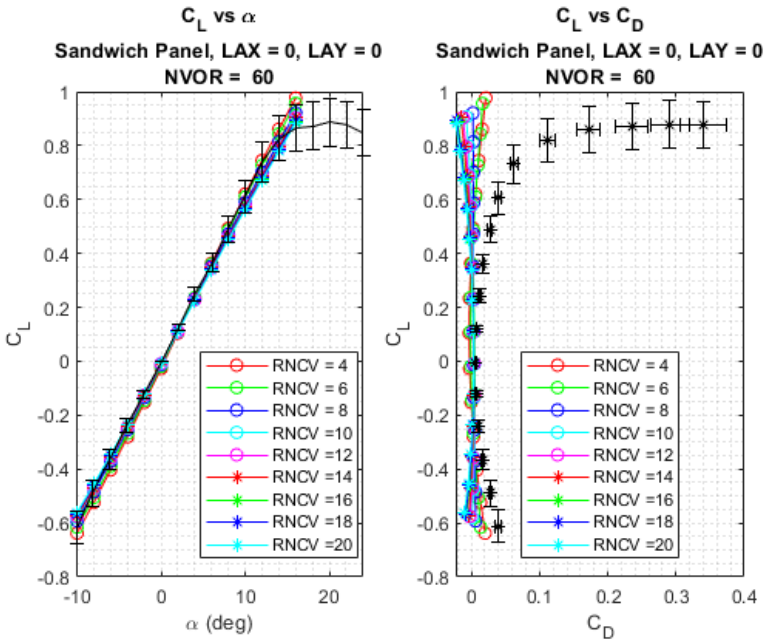


Figure 26c: NVOR and RNCV Comparisons for Sandwich Panel with Cosine Spacing, NVOR = 60

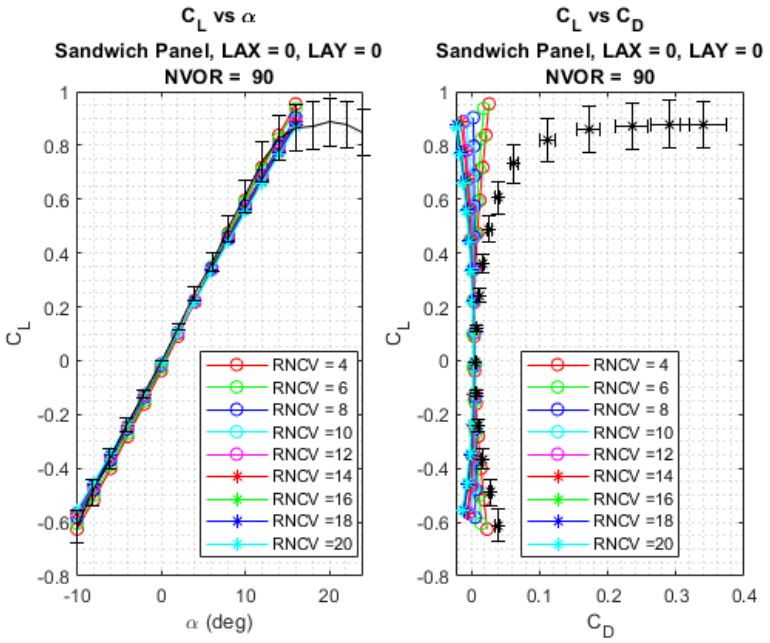


Figure 26d: NVOR and RNCV Comparisons for Sandwich Panel with Cosine Spacing, NVOR = 90

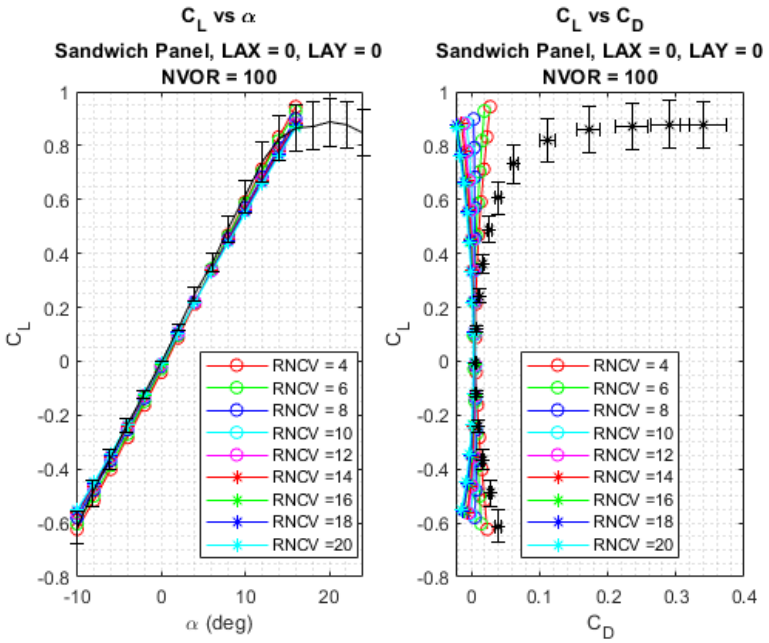


Figure 26e: NVOR and RNCV Comparisons for Sandwich Panel with Cosine Spacing, NVOR = 100

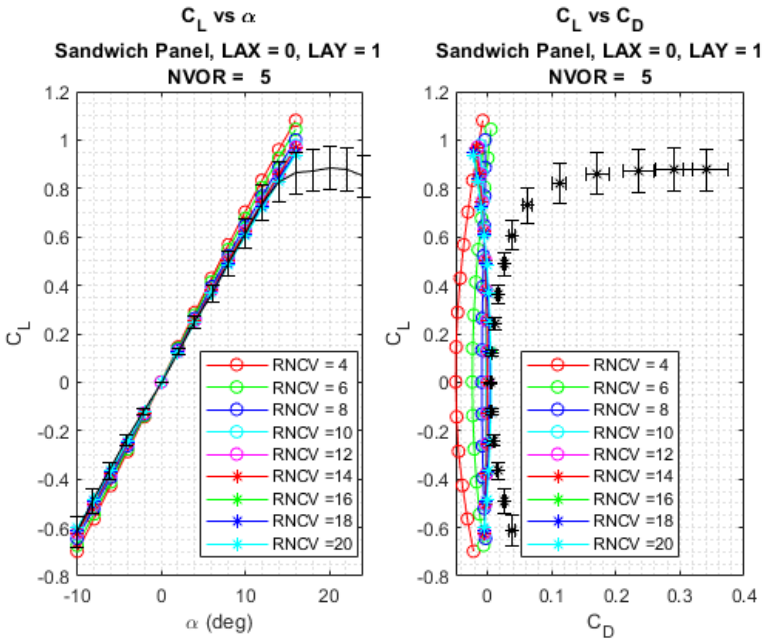


Figure 27a: NVOR and RNCV Comparisons for Sandwich Panel with Linear Spacing, NVOR = 5

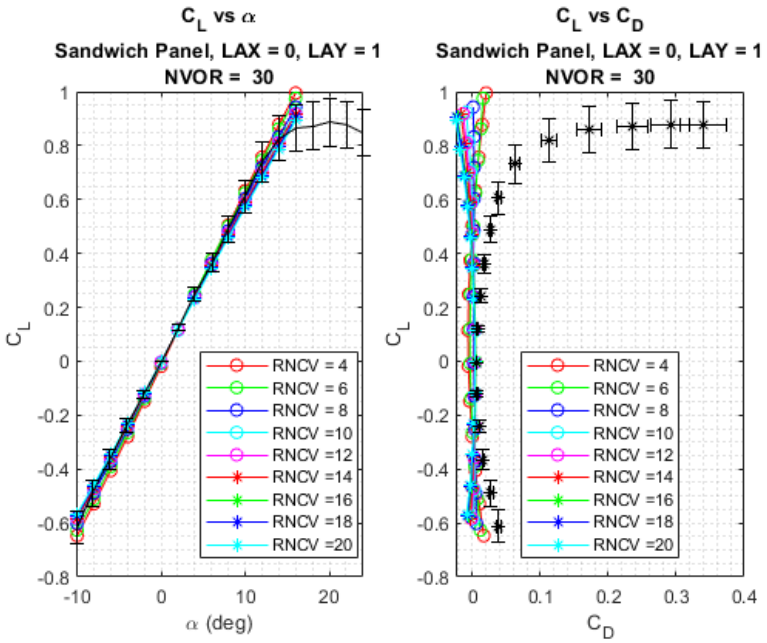


Figure 27b: NVOR and RNCV Comparisons for Sandwich Panel with Linear Spacing, NVOR = 30

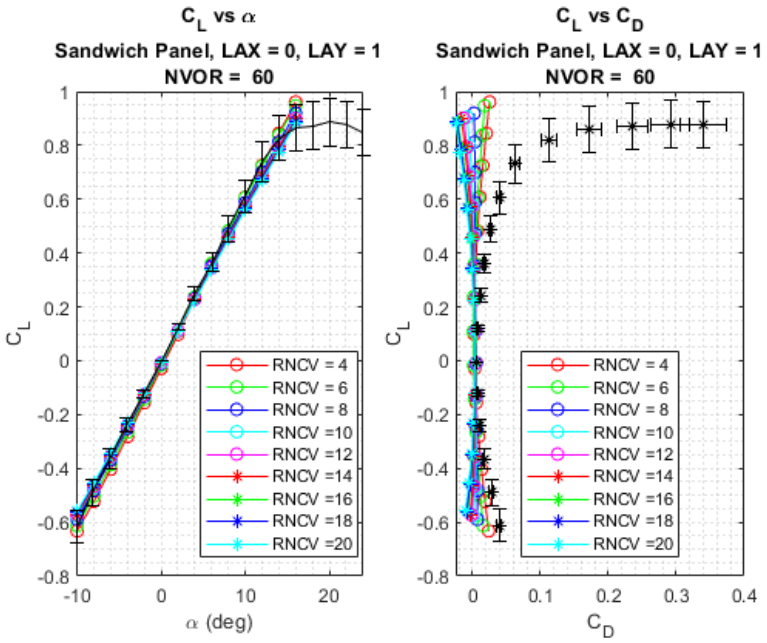


Figure 27c: NVOR and RNCV Comparisons for Sandwich Panel with Linear Spacing, NVOR = 60

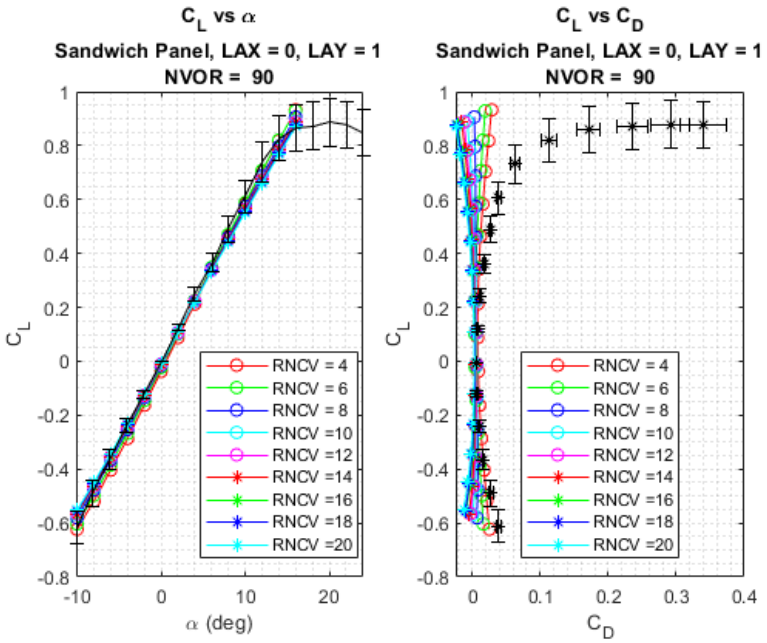


Figure 27d: NVOR and RNCV Comparisons for Sandwich Panel with Linear Spacing, NVOR = 90

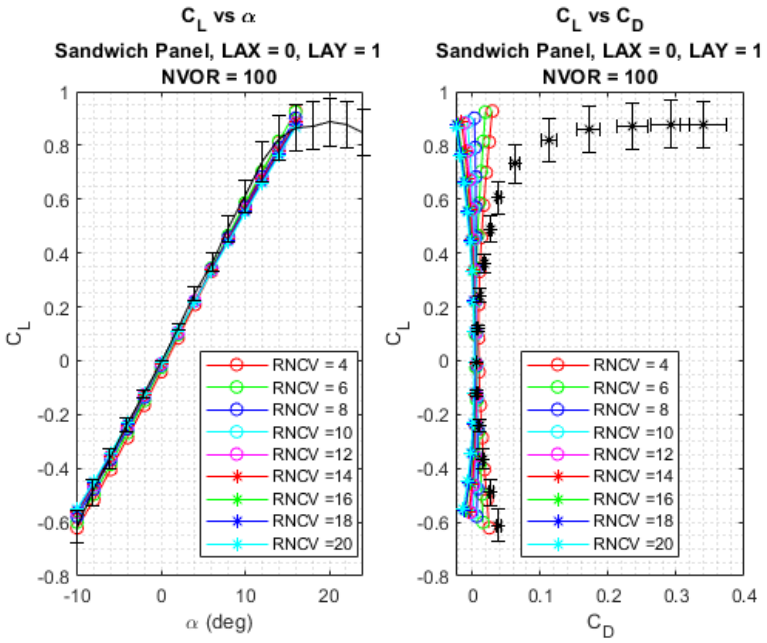


Figure 27e: NVOR and RNCV Comparisons for Sandwich Panel with Linear Spacing, NVOR = 100

Figures 26 and 27 show the results of the testing for the planar sandwich panels. From these plots, it is clear that while the lift curves are more-or-less spot-on relative to the tests, the drag polar plots have a lot more going on. Many of these solutions are blatantly incorrect. With the drag polar behavior, there is a progression from the plots which are close in magnitude, yet still incorrect, to values that point the complete opposite direction; this depends upon the density of the chordwise control points. Thus, for the planar sandwich panels with cosine spacing, the best correlation for drag is seen at $NVOR = 100$ and $RNCV = 4$ with an L2-Norm of 0.0153, however the best correlation for lift is at $NVOR = 10$ and $RNCV = 10$ with an L2-Norm of 0.0265, which gives completely incorrect drag values! Likewise, the linear spanwise spacing gives the best C_L correlation at $NVOR = 10$ and $RNCV = 10$ with an L2-Norm of 0.0265, and the best C_D correlation occurs at $NVOR = 100$ and $RNCV = 4$ with an L2-Norm of 0.0138. Thus, this method is suitable for lift measurements, but lacks the ability to properly resolve the drag coefficients. Furthermore, it should be clear from the above figures that the correlation with C_L remains good for almost all of the cases.

For years, students of Professor Takahashi have used *VORLAX* in its sandwich panel mode, but often found that the drag results were incorrect and as such they were neglected in studies. However, the sandwich panel configurations were used for detailed lift coefficient and pressure coefficient analysis. The most dedicated of the students would take advantage of the easy automation seen for *VORLAX* and would develop codes to run both a flat panel case and a sandwich panel case, giving them accurate lift and drag figures while also providing accurate pressure distribution data.

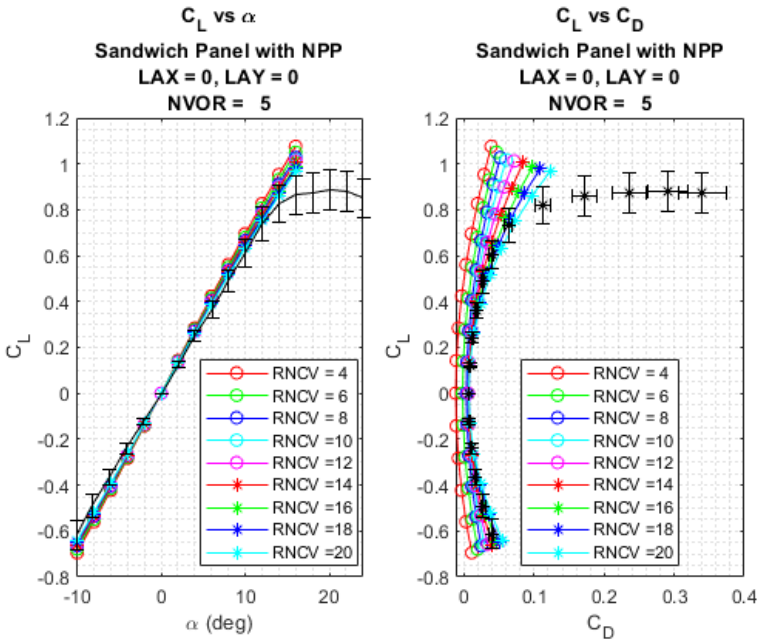


Figure 28a: NVOR and RNCV Comparisons for Nonplanar Sandwich Panel with Cosine Spacing, NVOR = 5

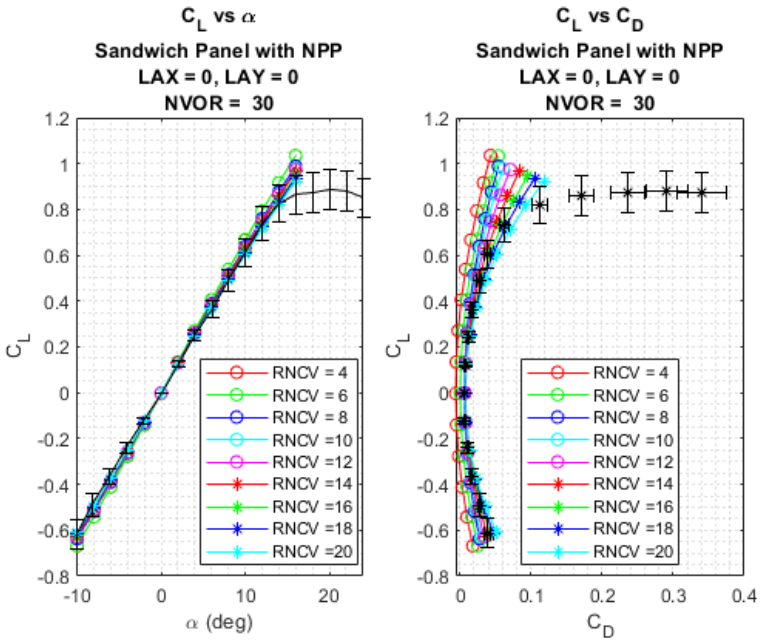


Figure 28b: NVOR and RNCV Comparisons for Nonplanar Sandwich Panel with Cosine Spacing, NVOR = 30

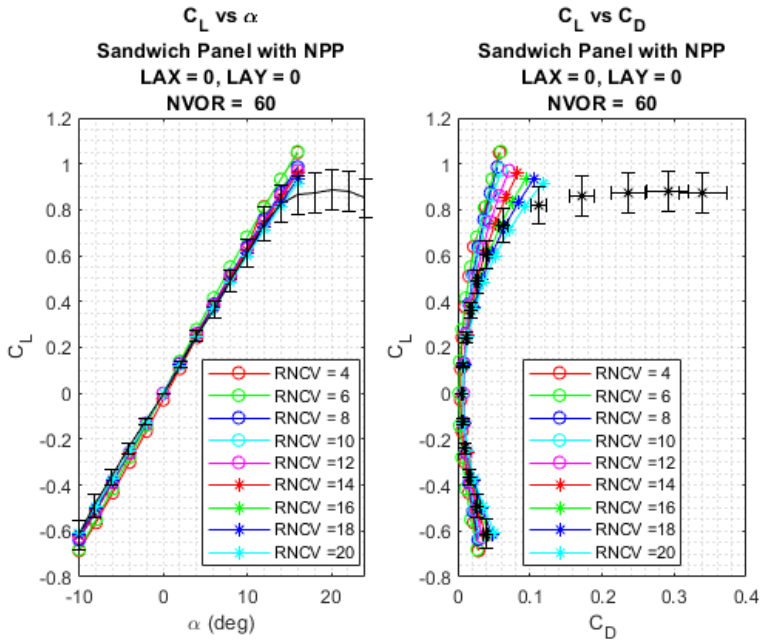


Figure 28c: NVOR and RNCV Comparisons for Nonplanar Sandwich Panel with Cosine Spacing, NVOR = 60

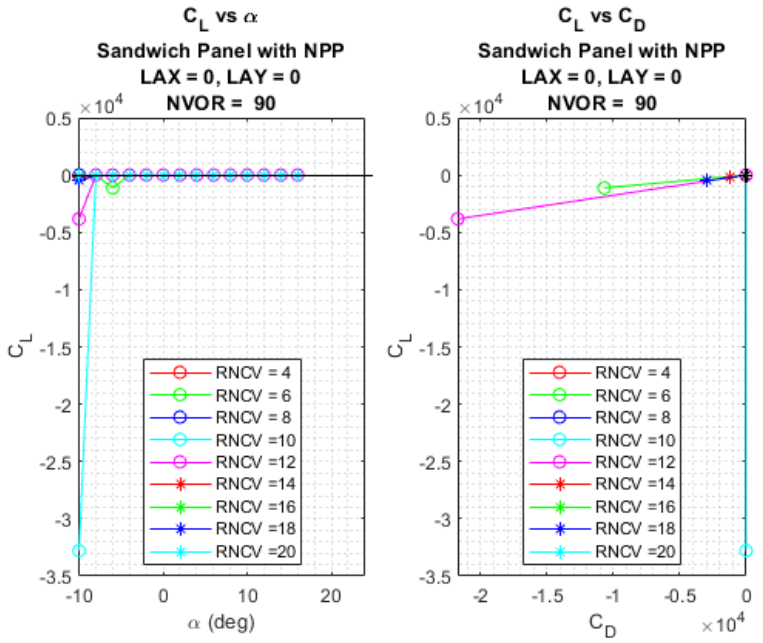


Figure 28d: NVOR and RNCV Comparisons for Nonplanar Sandwich Panel with Cosine Spacing, NVOR = 90

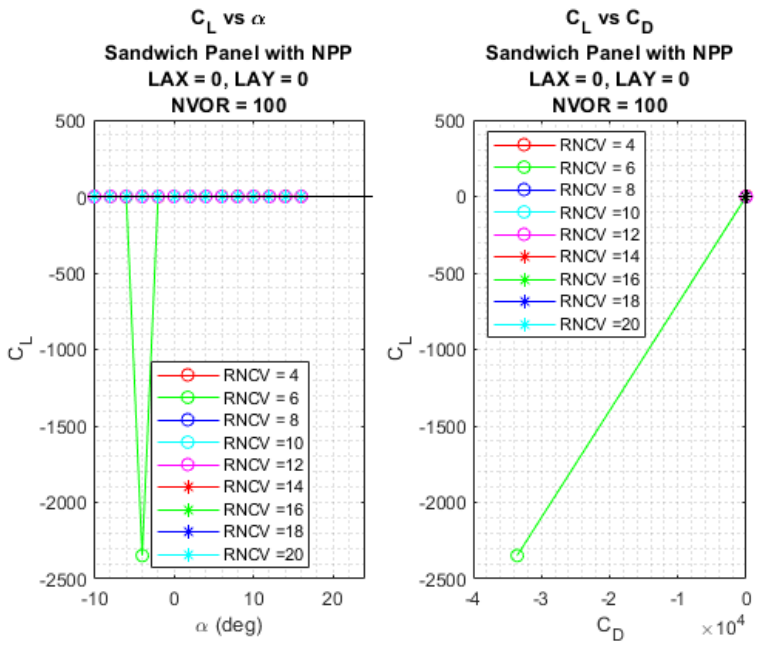


Figure 28e: NVOR and RNCV Comparisons for Nonplanar Sandwich Panel with Cosine Spacing, NVOR = 100

The *VORLAX* nonplanar sandwich panel gives fantastic results when configured correctly, alleviating the need to run alternate models alongside the primary model of interest. Figure 28 shows the correlation between the nonplanar sandwich panel and the test data. Recall that while the planar mode will hold the control points at a single location in the z-direction, the nonplanar mode will apply the geometric perturbations from thickness and camber, which theoretically should be more accurate. When running the tests, this was found to be the case, where both the lift and drag figures showed fantastic agreement for many of the configurations.

Unfortunately, there are a few cases that diverge aggressively, giving incredibly incorrect answers. These are seen in the latter two plots, Figures 28d and 28e. While poor results are not acceptable, it should be noted that the errors in this case are so bad that it is improbable that a user would blindly use these as their basis for design. When parsing the code in search of what may cause these errors, it was found

that the nonplanar parameter appears in only two places. In one subroutine, a small angle approximation is made for planar panels, while nonplanar panels have exact trigonometric quantities calculated. A short test was run which deemed this was not the problem. The only other place that it appears is in the geometric building routine, where it predictably tells the program to displace the control point locations. Thus, presently these errors persist, and the error appears to be a side effect in other subroutines of the program, rather than an error imposed directly by the nonplanar panels.

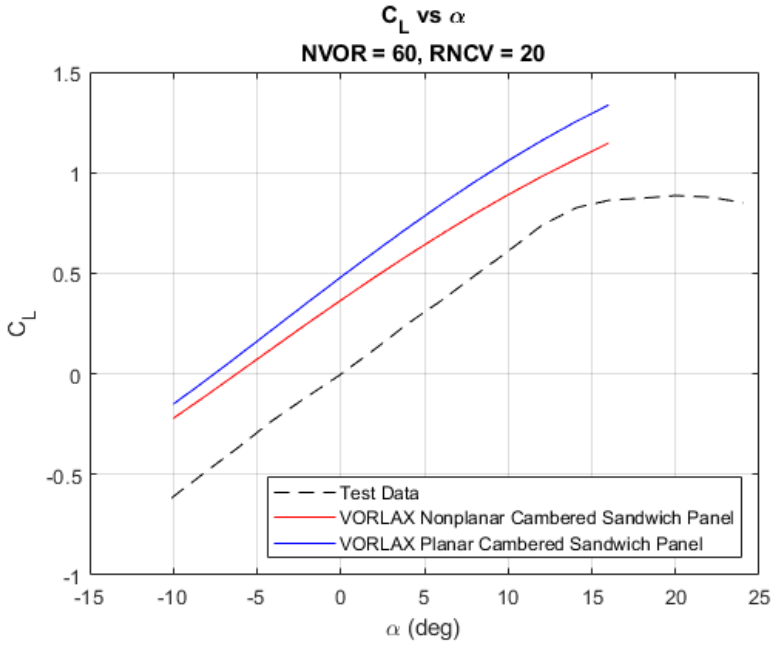


Figure 29a: Nonplanar and Planar Sandwich Panel Lift Coefficients

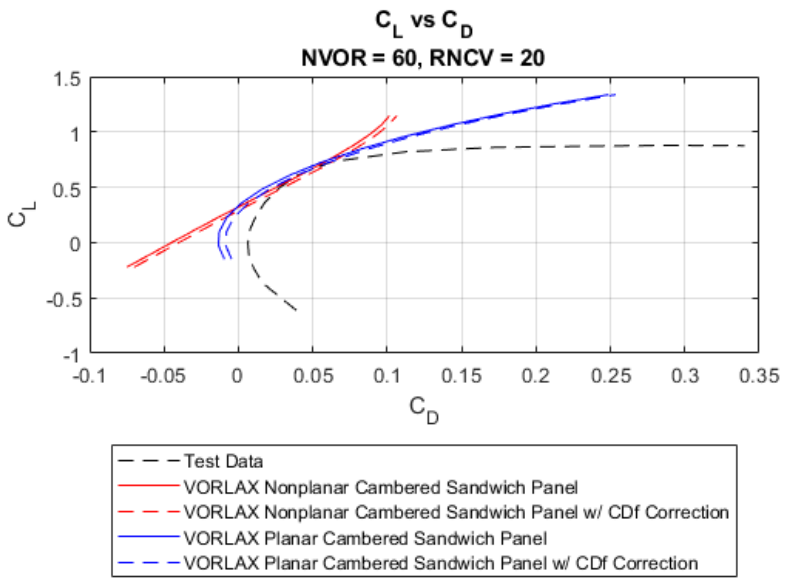


Figure 29b: Nonplanar and Planar Sandwich Panel Drag Coefficients

Figure 29 shows the results of the thick sandwich panel with the addition of camber in the planar mode. In this case, there is the familiar behavior with the addition of camber, showing a linear increase in lift coefficient for all pitching angles. Similarly, there is the familiar change in the drag polar, seen in the earlier flat panel cambered case. However, running the nonplanar test case demonstrates different behavior, which was not the case for the symmetric panel. The results returned by the nonplanar mode show a lesser increase in the lift coefficient due to camber, which in turn affects the drag polar. This begs the question, which is correct?

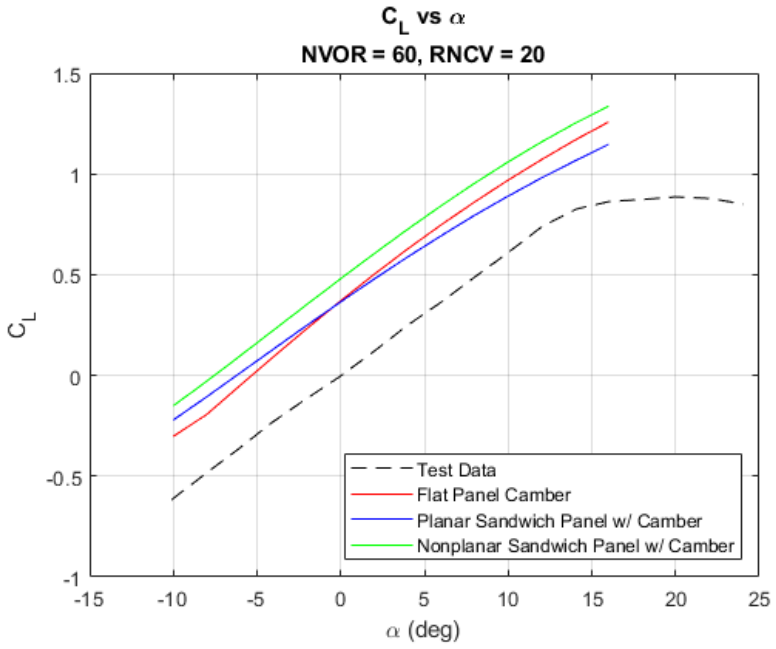


Figure 30: Verification for Panel Planarity

Prior work has shown that it is reasonable to apply the principle of superposition to account for the change in lift coefficient arising from thickness, camber, and wing twist [21]. Thus, to reconcile the difference and determine which mode was correct, an additional plot was made to apply the increase in lift due to camber alone with the lift curve for the symmetric sandwich panel. Figure 30 shows that the thick panel with camber agrees with the flat panel case and recall that both the planar and nonplanar cambered cases were nearly identical. Thus, the nonplanar case of a cambered sandwich panel incorporates an incorrect constant increase in lift relative to the planar case.

Within the *VORLAX* code, there are a few places where small-angle approximations appear. One of these was tied to the nonplanar parameter (*NPP*). In an attempt to reconcile the difference between the planar and nonplanar case, a value for *NPP* was used to avoid the small-angle approximations. Doing so generated lift coefficients

much closer to the expected values, however there were still large discrepancies in the drag polar figures. This implies that there is more to the difference than simply the small angle approximation. However, the nonplanar parameter does not actually appear often in the *VORLAX* source code, and so the differences in the drag may be arising due to conflicts in other portions of the code, such as those related to the induced drag. Because the linear chordwise spacing was shown to significantly impact the quality of the drag polar measurements, there is a chance that the displacement of the points due to nonplanarity may lead to errors, as well.

Thus, it is conclusive that for the case of a cambered panel with thickness, the nonplanar option is not viable. In some cases mentioned above, the errors were drastic and obvious, which is preferable. When using *VORLAX*, it is very easy to see $C_L = 35,000$ and know that something must be incorrect, but in this case the lift coefficients are incredibly reasonable and believable, but inconsistent with the expected values. This can be particularly damaging when using *VORLAX* as a means of first-order design considerations and approximations. Thus, it is inadvisable to rely on the nonplanar case for wing design. Most practical design cases will involve the designer incorporating camber into the wing, invalidating the results from the nonplanar case.

The nonplanar case will be included in the compile of the program due to two considerations. The first is that most students do not know what it is and as such they do not use it. In the event that the students do become adventurous and attempt to use it, there will be a disclaimer asterisk printed to the LOG and output files. Likewise, those who use the tool professionally have already come to terms that the method is not reliable and do not use it. While this introduces an unlikely risk that

someone will break their model, it is useful for visualization reasons, further outlined in Chapter 6.

When determining the ideal grid spacing for the nonplanar configurations, it became apparent that the mode had some more nuance necessary for its operation than the planar version did. There were some cases where the skewedness of the grid presented problems for the model, presenting values that were drastically incorrect. The errors were seen for small $RNCV$ values (such as those less than 10) and large $NVOR$ values (such as those greater than 70). The MATLAB correlation test was run again with the nonplanar solution to determine the ideal grid spacing for the symmetric case. Running the analysis on the grid spacings, the lift curve had best agreement with $NVOR = 80$ and $RNCV = 10$, while the drag curve had best agreement with $NVOR = 5$ and $RNCV = 16$. Nonetheless, this was an experiment of curiosity, as the $NPP = 1$ mode is considered unusable in its current state.

From the above tests, the value of $NVOR \approx 60$provides consistently good results, as does the value of $RNCV \approx 20$ when paired with the cosine spanwise spacing. This density serves as the middle ground between those cases which favored the lift coefficient and those which favored the drag coefficient. Nonetheless, even if the chosen benchmark density did not provide the best results for a case, this density was reasonable for all cases with the flat panel. This is consistent with anecdotal evidence and prior experience indicating that "about 100 total spanwise points" is a good value for $VORLAX$ models. Thus, for the ensuing pressure contour tests, this configuration was run. While not "the best" for some of the cases, the small amount of error introduced by a "non-ideal" grid is inconsequential within the scope of assumptions under which $VORLAX$ operates.

The results of the grid density testing are very complimentary to the intended

usage case for *VORLAX*. This program is meant to be something that an engineer may use on their work machine without a ton of preprocessing work or library updates. It is meant to work efficiently, and that begins at the human operating the program. The results returned are enough to help the user estimate the lift of a configuration, estimate the drag of the configuration, as well as other things such as the stability and elliptical loading of the design. While the above outlines the scenarios for "ideal" grid configurations, it is important to note that almost none of the configurations give results that are completely incorrect to the point of destroying intuition. For those configurations that do give remarkably poor results (namely those with the sandwich panel method), the results behave in a manner where any engineer with an understanding of aerodynamics should be able to identify that they are incorrect. That is, in essence, the "point" of the Vortex-Lattice Method – it exists as an underappreciated tool that is immensely useful for first-order design considerations.

While it has been repeated that *VORLAX* cannot completely resolve the behavior of transonic flow, it can provide valuable insight to the nature of the wing overall, namely providing information that an engineer can utilize to obtain an elliptical flow distribution over the main wing of an aircraft by means of wing twist, camber, and thickness. To do so, it is necessary to use one of the more complex methods described above, as the flat panel cases cannot, by definition, represent the thickness and camber. However, they can be used for a superficial test of basic wing twist – a test which can be trivially automated via a design software such as ModelCenter, or even through a simple spreadsheet.

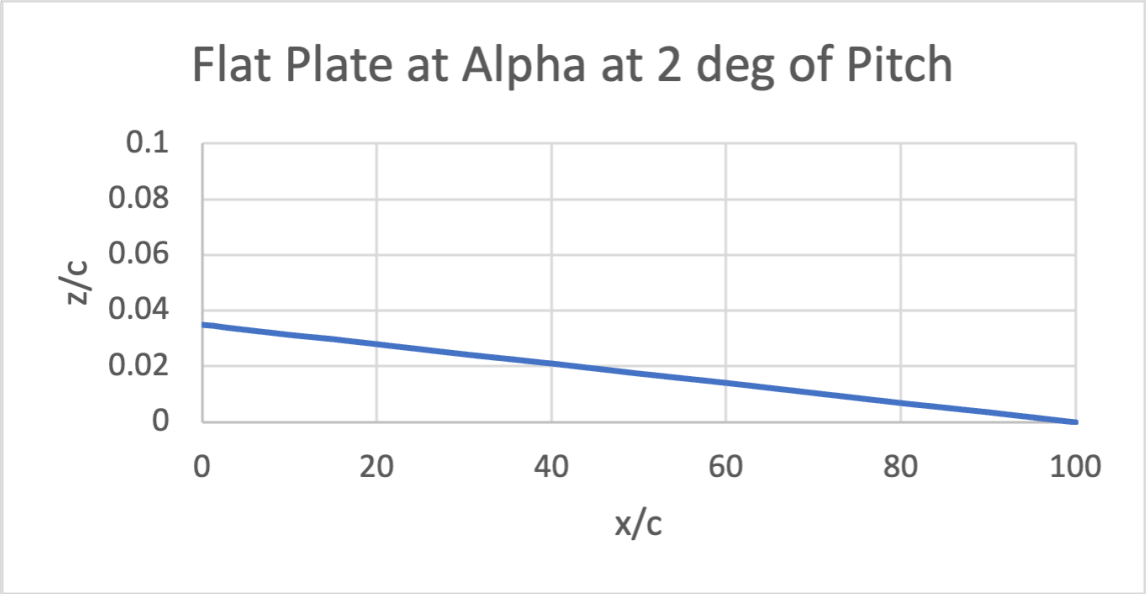


Figure 31: Flat Panel Profile [9]

Figure 31 shows a flat panel inclined at 2 degrees angle of attack, showing just how simple this representation of an aerodynamic surface is. Because the Vortex-Lattice Method is concerned more with the influence at each control point due to each of the bound vortices, simple representations like the flat panel work as they can offer a fairly reasonable "main idea" of the slope of a surface, which is then used to compute the bound vortex strength. Comparably, a finite-difference approach uses only the neighboring points and thereby requires a much more refined model to return accurate results. Likewise, the Vortex-Lattice Method allows the user to utilize a relatively low number of grid points, however some parameters are resolved via numerical integration, in which case the values may stray from "reasonably correct" should the grid density become too low.

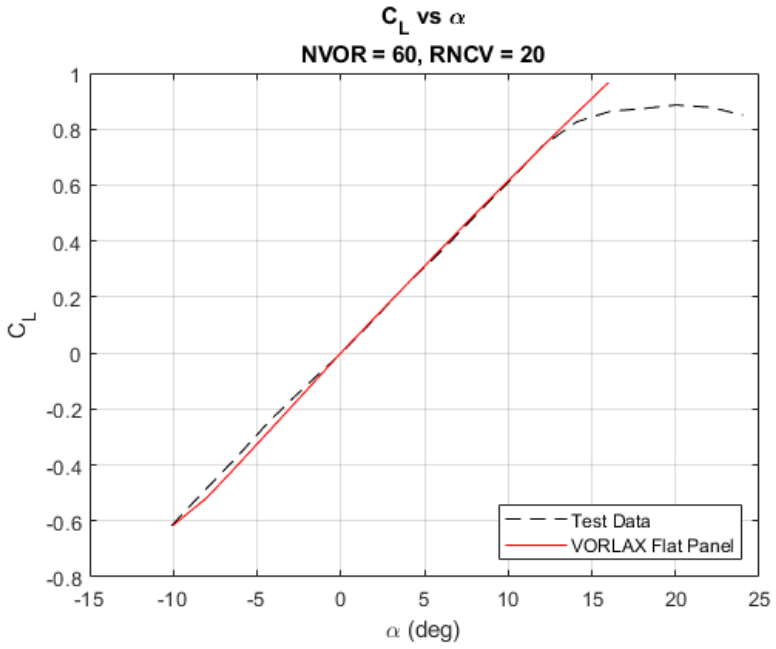


Figure 32a: Lift Curve for Flat Panel

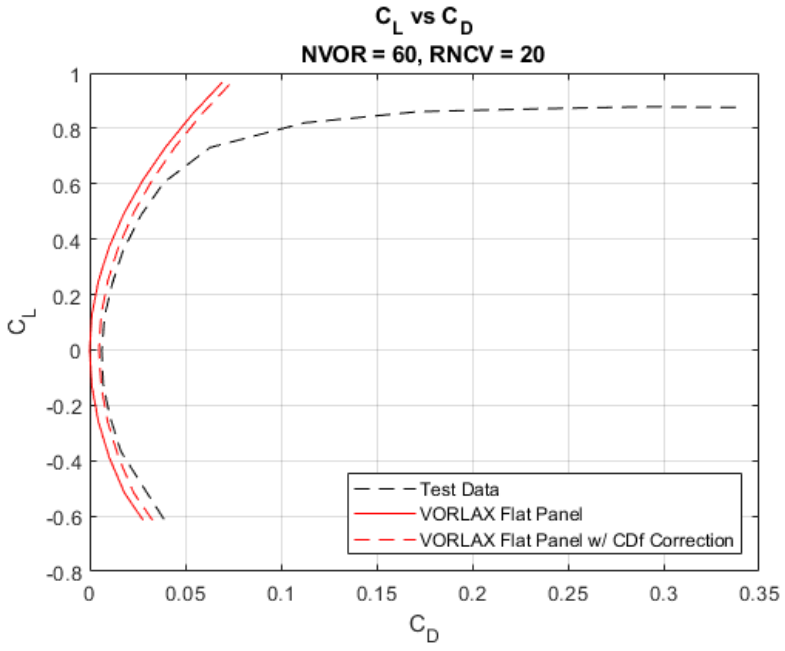


Figure 32b: Drag Polar for Flat Panel

Figure 32 shows the comparison of the flat panel data to the test data when run at a single grid density. Assuming that a 10% error tolerance is acceptable, such as in the case of a preliminary design application, the *VORLAX* lift curve fits incredibly well, especially for positive angles of attack. In the region where the magnitude of alpha is small, the line aligns almost perfectly with the true wind tunnel value. Furthermore, Figure 32 shows that at these same small angles of attack, the values returned by *VORLAX* also align almost perfectly. Within the *VORLAX* source code, there are occasional approximations made for cases such as those with a small angular displacement, thus it makes sense that they will return similar values. As the magnitude of alpha grows, the values "fan out" and become more unique, though none stray far enough from the mean that they would indicate incorrect results. In fact, for the 10% tolerance shown by the error markers, almost all of the lines fall within the window in the stall-free region.

There is a similar behavior with the drag polar, where all the *VORLAX* values are reasonably in-line with one another, however, there is a noticeable discrepancy in the drag coefficient magnitudes near the low angles of attack. This is expected, given the assumption by *VORLAX* that the flow is inviscid. As the angle of attack increases, the relative contribution of the pressure forces exceeds those of the viscous forces, and so as the relative importance of the viscous terms of the overall drag decrease, the result given by *VORLAX* approaches that of the real-world wind tunnel test. To improve this correlation, a model using a basic form-factor equation [8] was used to provide a reasonable estimate of the friction drag on the half-span wing.

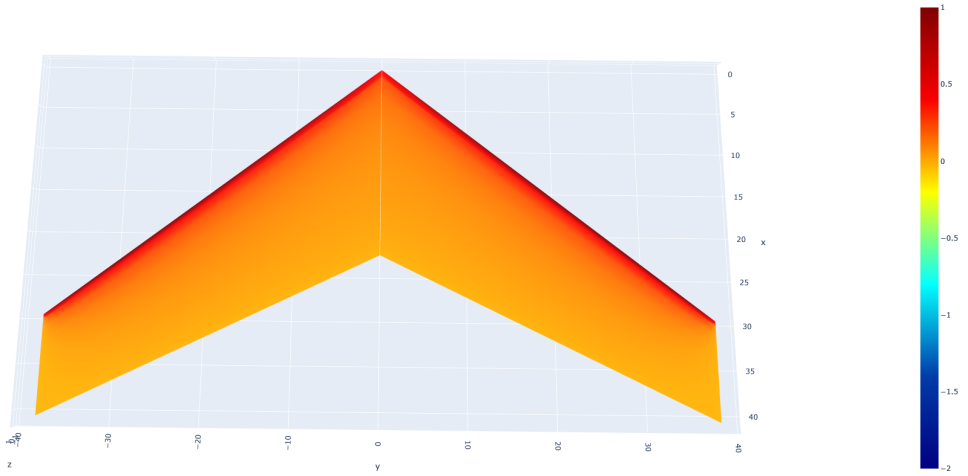


Figure 33: Flat Panel Pressure Contour

When the interest shifts from macroscopic aircraft performance parameters to more nuanced parameters, the shortcomings of the flat panel mode come to light. The contour of the loading patterns on this example wing shows that the flat panel can only provide so much information. In the case of the flat panel, this plotted quantity "DCP" represents the pressure differential between the lower and upper surface, where positive DCP indicates a positive lift. This contour shows that over the surface of this entire half-span wing, the pressure magnitudes are roughly the same. This is not terribly inaccurate, as the plotted quantity is essentially equal to

$$DCP_{\text{LOWER}} - DCP_{\text{UPPER}}$$

, and there are no provisions within the code that allow the user to break these apart. There are some differences on the surface, with a slight negative gradient moving aft along the chord, but nothing that would be helpful when working through a shock-free wing design process. Looking at this plot, it lacks any useful information regarding the local peak pressure magnitudes and where they may be located. Thus, while there is

clear information showing a net positive value of lift, the lack of localized information makes this method inadequate for an advanced wing design.

5.3 The Cambered Panel

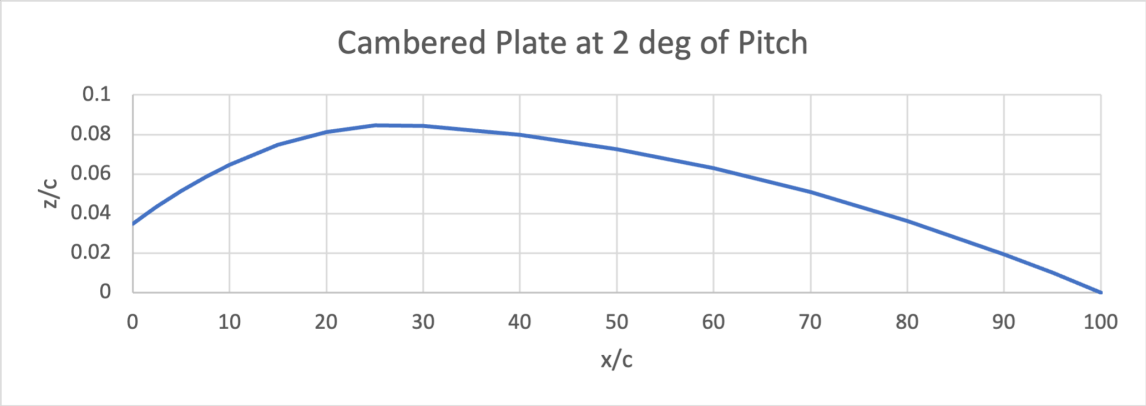


Figure 34: Cambered Panel Profile [9]

The cambered panel is another flat-panel representation of an aerodynamic surface within the *VORLAX* program. This mode remains geometrically simple and still relies on the resolution of a zero-mass flux condition on each side of each panel, but it provides the user with slightly more insight as the flat case. Figure 34 shows the shape of a NACA 63 mean line, however this is only one of the ways that *VORLAX* may represent this shape. Figure 34 shows the model of a flat panel with camber in the event where the nonplanar parameter, NPP, is set equal to one, thereby having the panel displaced in space, rather than manipulating only the slopes in a planar fashion.

The other (more common) method of drawing a *VORLAX* panel is when the panel is treated as planar, with $NPP = 0$. When the panel is planar, the perturbations

to the shape from thickness and camber effects are represented only by their slopes, and therefore lack any actual displacement in their location as seen by *VORLAX*. For rather thin panels, or perhaps those without any extreme camber profiles, this works fine, However, as the magnitudes of these displacements grow, this approximation may not be reasonable.

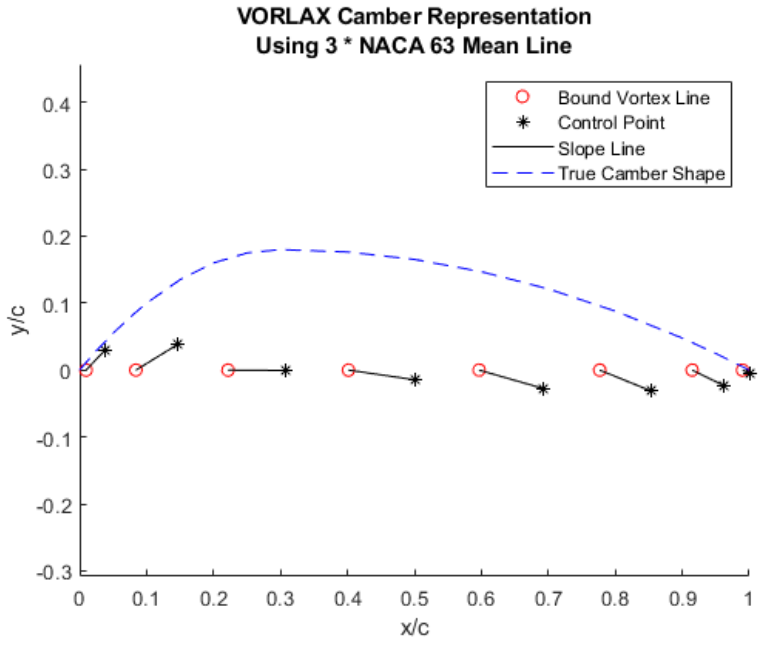


Figure 35: Cambered Panel Slopes [9]

The original reference study featured a geometry lacking camber. Nonetheless, it was expected that the flat panel configuration would show a constant increase in the lift coefficient due to the addition of camber. Upon testing the two versions and comparing them to the flat plate test case, the curve showed the expected behavior, following the same slope as the uncambered case, except with a higher overall lift coefficient. This was the case with both the planar and nonplanar configurations – there was no substantial difference between the two methods.

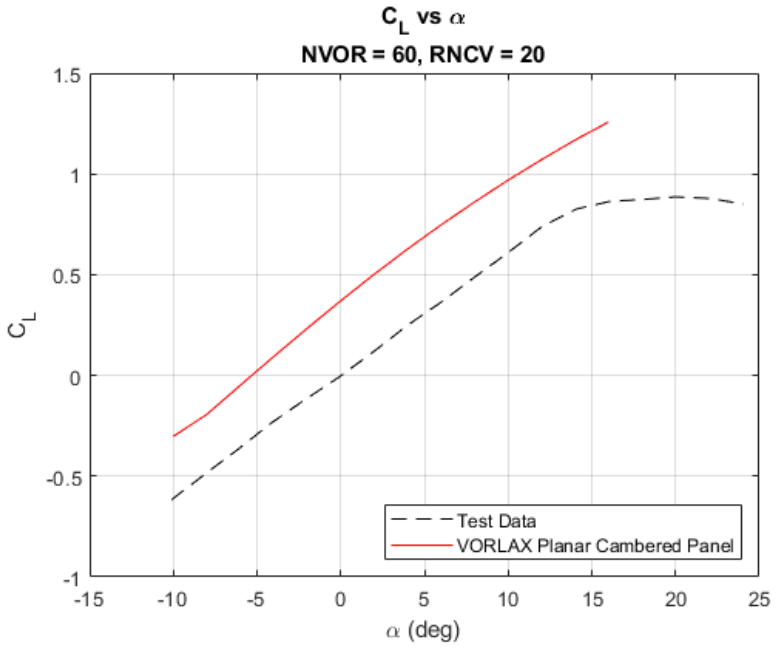


Figure 36a: Lift Curve for Planar Cambered Panel

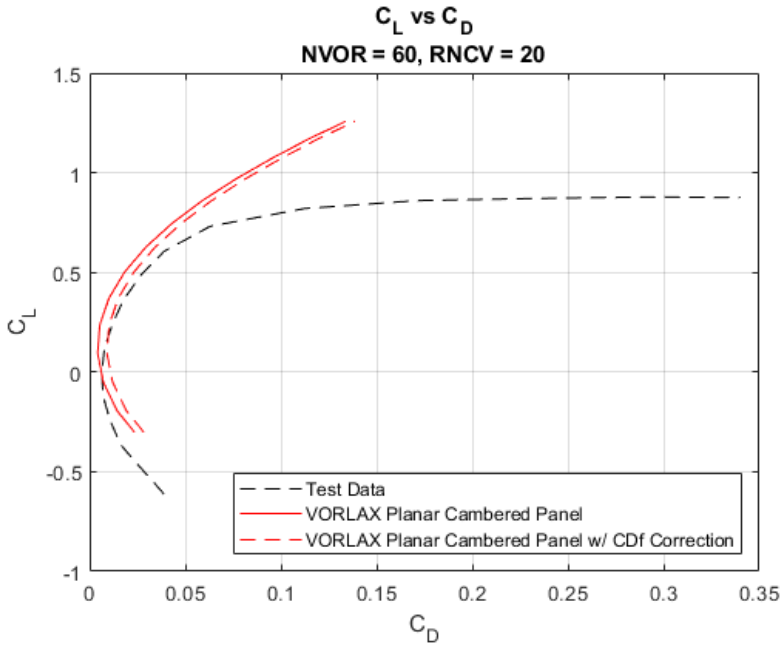


Figure 36b: Drag Polar for Planar Cambered Panel

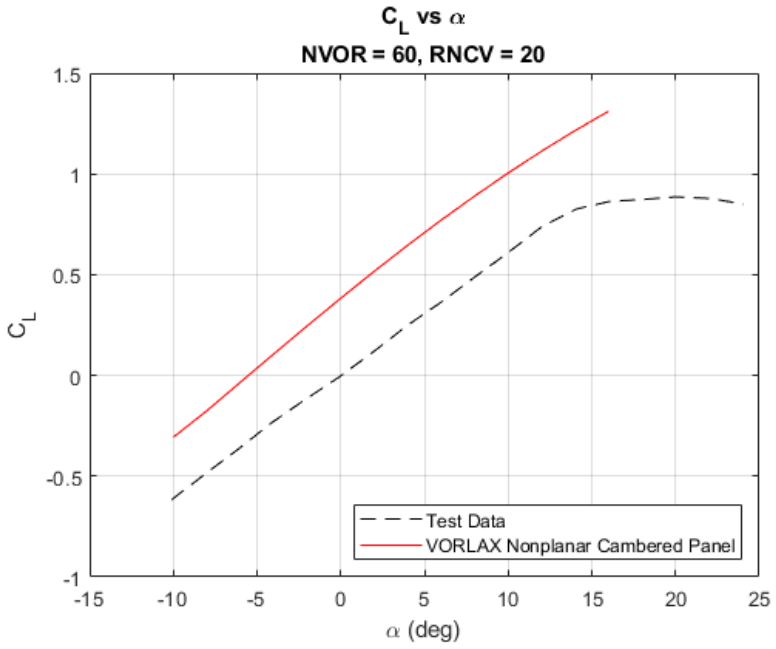


Figure 37a: Lift Curve for Nonplanar Cambered Panel

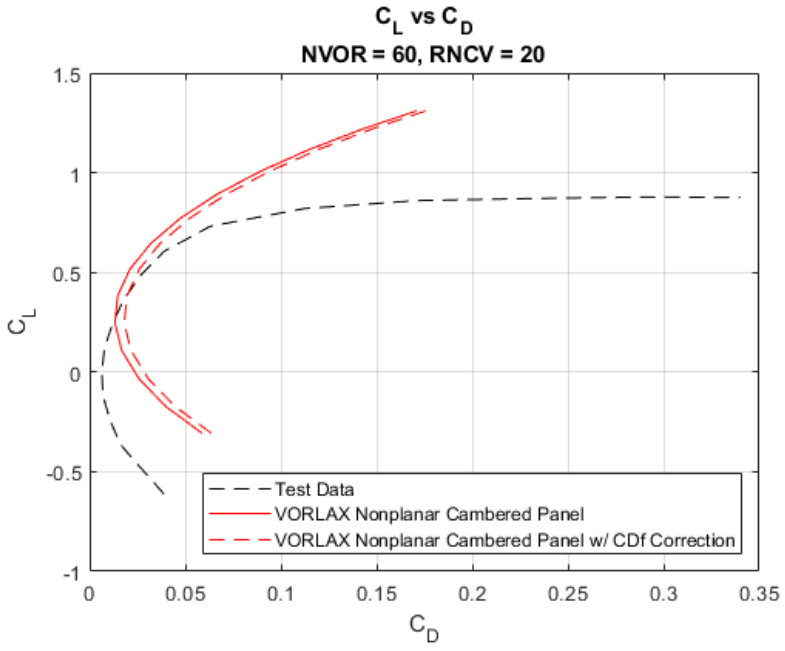


Figure 37b: Drag Polar for Nonplanar Cambered Panel

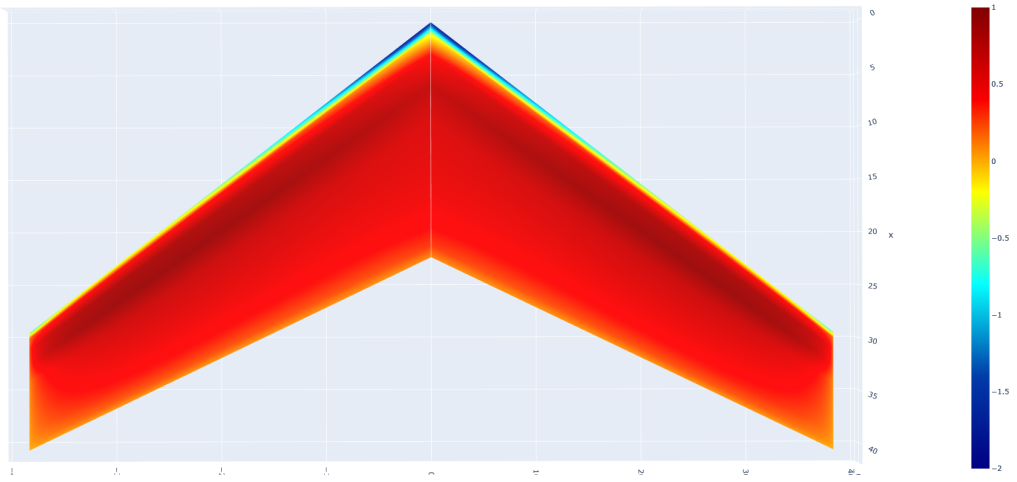


Figure 38a: Contour for Planar Cambered Panel

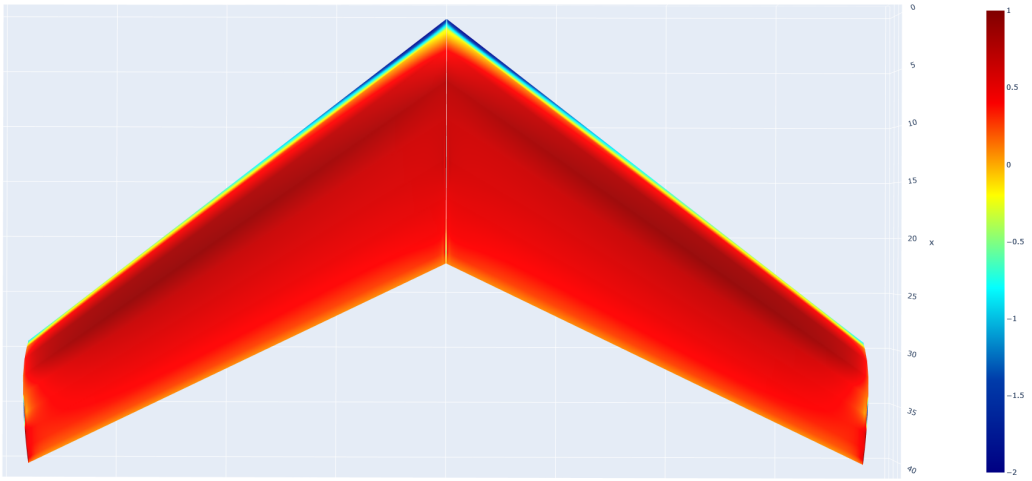


Figure 38b: Contour for Nonplanar Cambered Panel

5.4 Thick Sandwich Panels

The natural progression for further wing fidelity is to look into including finite thickness effects for analysis. To simulate a panel of finite thickness, the user will use two panels and place them some small distance apart, parallel to one another. This small distance was specified by Miranda in the original theory guide, where an overall spacing of 2/3 the maximum distance at the spanwise station was advised [1]. Thus, the user will incorporate this with small changes in the z-coordinates for both the upper and lower panels, with this value given as

$$z = \pm \left(\frac{1}{3}\right) \left(\frac{t}{c}\right)_{\text{MAX}} (y) \tag{5.1}$$

When considering a sandwich panel design, the user will also make changes to the zero-flux condition. With this usage scenario, the designer will enforce the zero-mass flux condition only on the outer surface via a flag in the panel definition card within the input file. This alters the linear system to be solved by changing the influence coefficients associated with each control point.

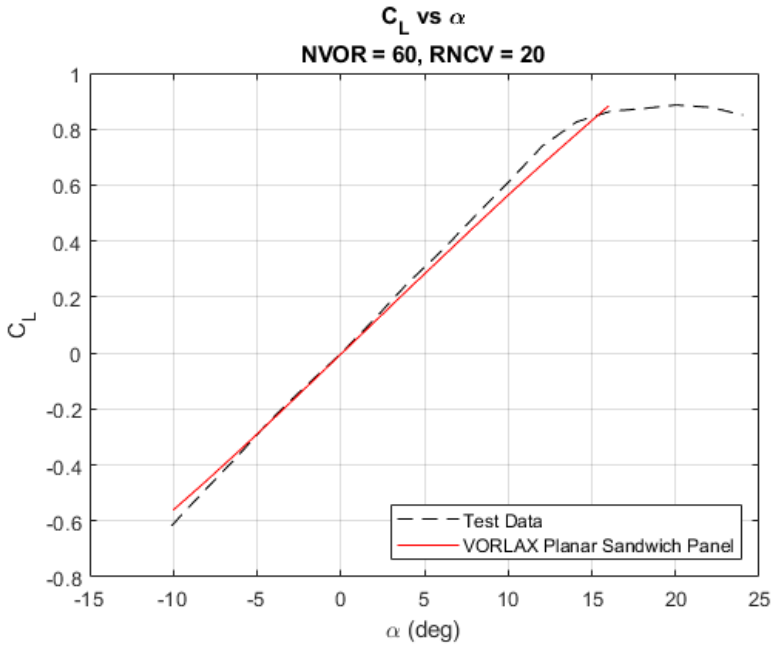


Figure 39a: Lift Curve for Planar, Uncambered, Sandwich Panel

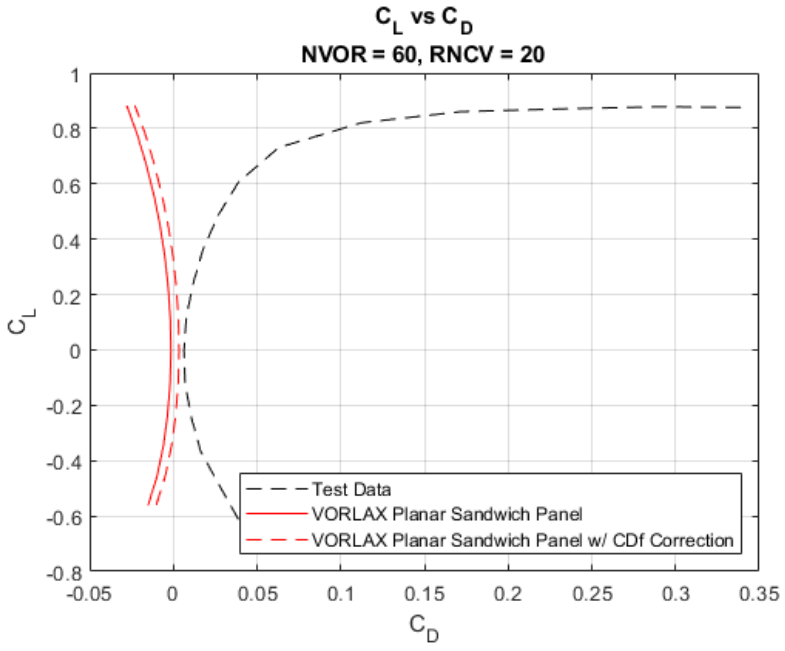


Figure 39b: Drag Polar for Planar, Uncambered, Sandwich Panel

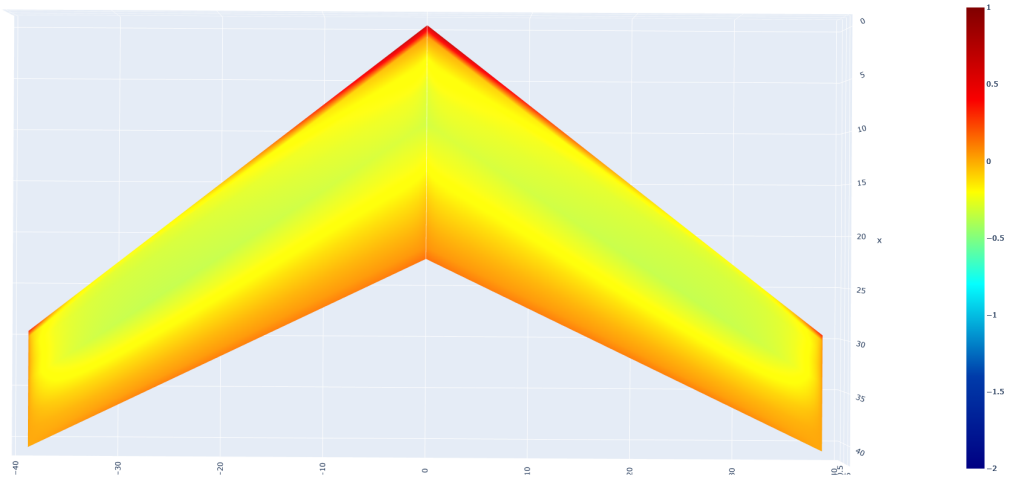


Figure 40: Pressure Contour for Planar, Uncambered, Sandwich Panel

Adding finite thickness to the *VORLAX* model increases the accuracy mainly when considering the local pressure coefficients. When a wing is thick, there are local flow accelerations (much like those for a cambered wing) that will lead to a more negative pressure coefficient. When using *VORLAX*, the user is often working on a transonic wing, attempting to design the surface as "shock-free", as such, the local variations due to thickness effects become significant. There are also some preferable additions arising from thickness, namely an increase in lift that mimics that of an increased freestream Mach number. As the wing becomes thicker, one may logically understand that the airflow will need to accelerate more to move around the wing, thereby increasing the potential pressure differentials when pitched.

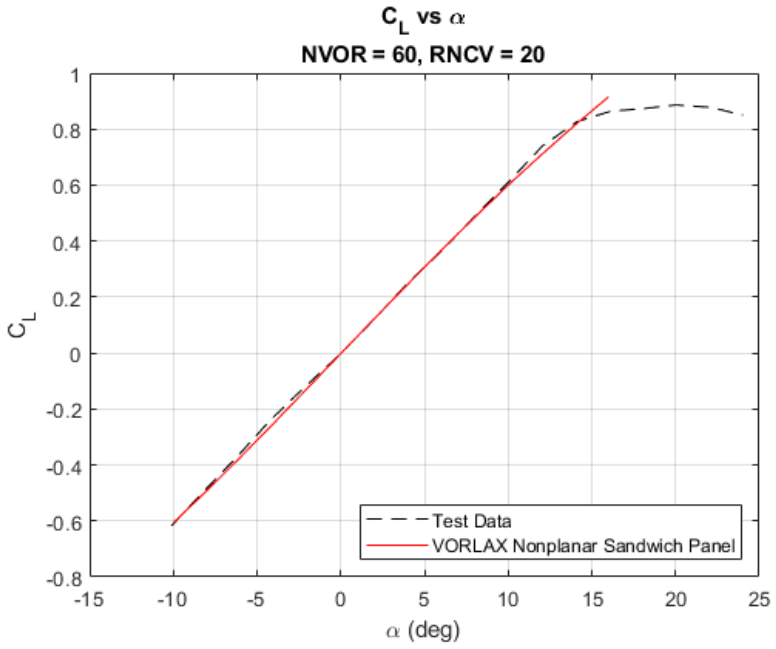


Figure 41a: Lift Curve for Nonplanar, Uncambered, Sandwich Panel

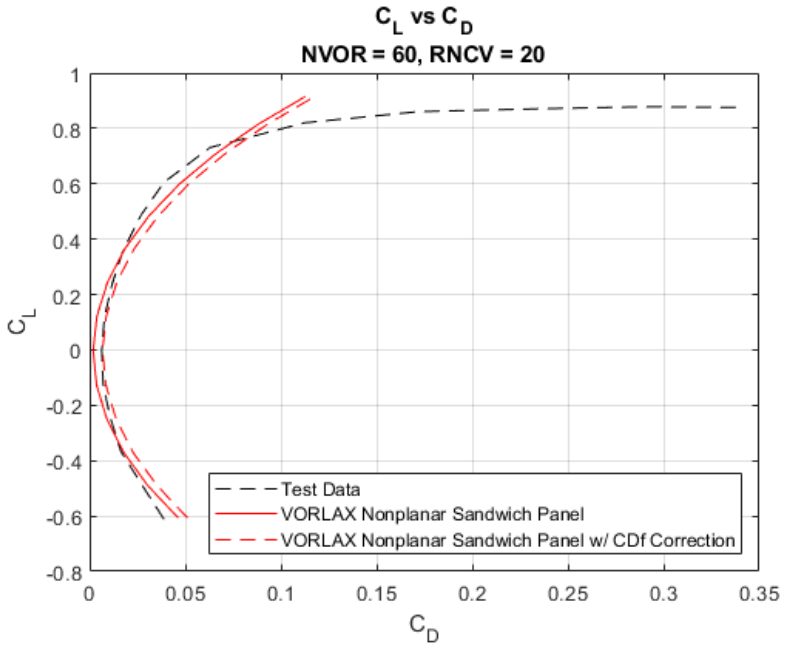


Figure 41b: Drag Polar for Nonplanar, Uncambered, Sandwich Panel

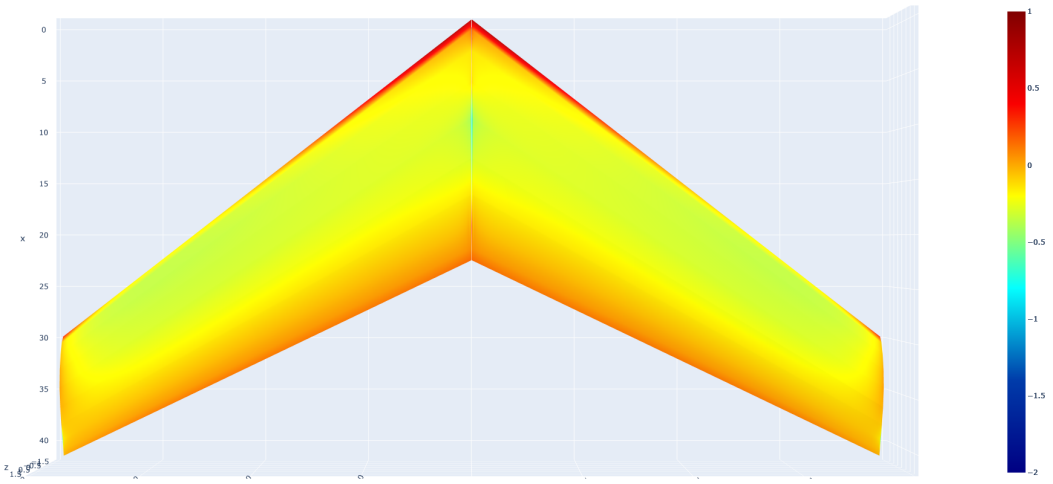


Figure 42: Pressure Contour for Nonplanar, Uncambered, Sandwich Panel

Much like the cambered panels, the sandwich panels have the option for both a planar and nonplanar representation. This also introduces the same considerations as in the cambered panel case, though now the usage of the NPP flag will alter the z-coordinates of the panels. Again, it should be noted that the $NPP = 1$ flag (nonplanar geometry) does not give accurate lift and drag results in the current state of *VORLAX*, however the method is compatible with the visualizer mentioned in Chapter 6, allowing the generation of 3-D nonplanar models.

5.5 Thick Sandwich Panel with Camber

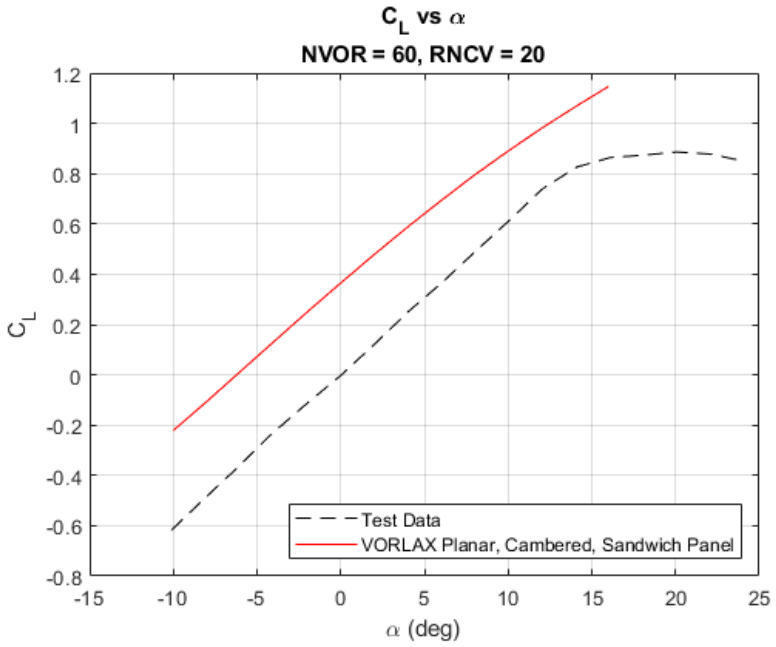


Figure 43a: Lift Curve for Planar, Cambered, Sandwich Panel

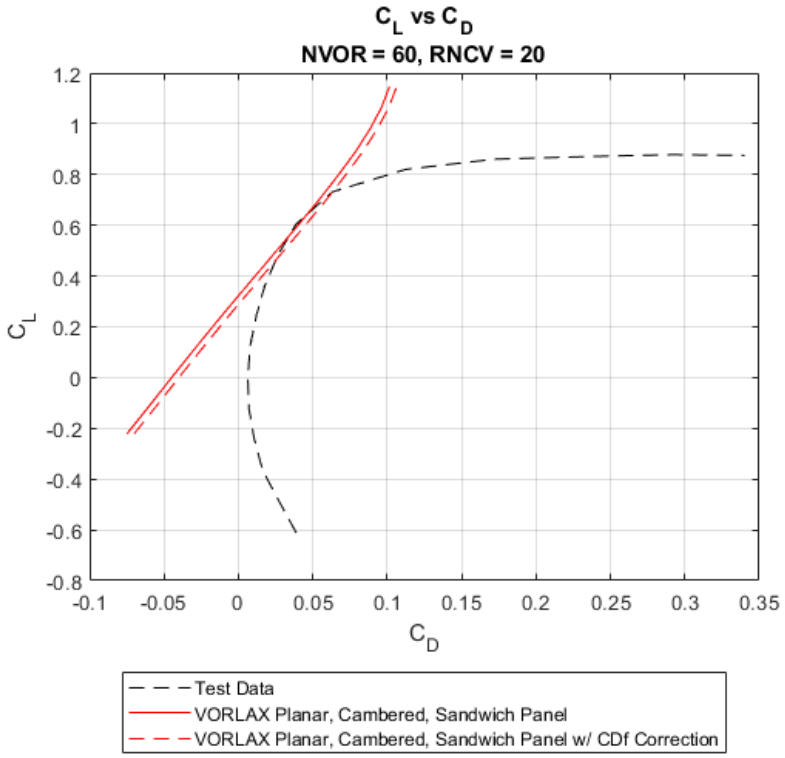


Figure 43b: Drag Polar for Planar, Cambered, Sandwich Panel

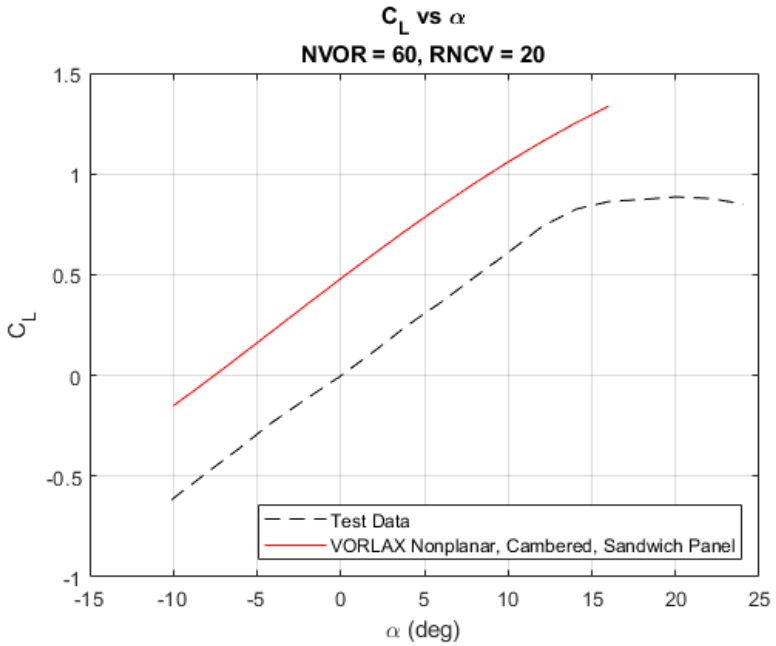


Figure 44a: Lift Curve for Nonplanar, Cambered, Sandwich Panel

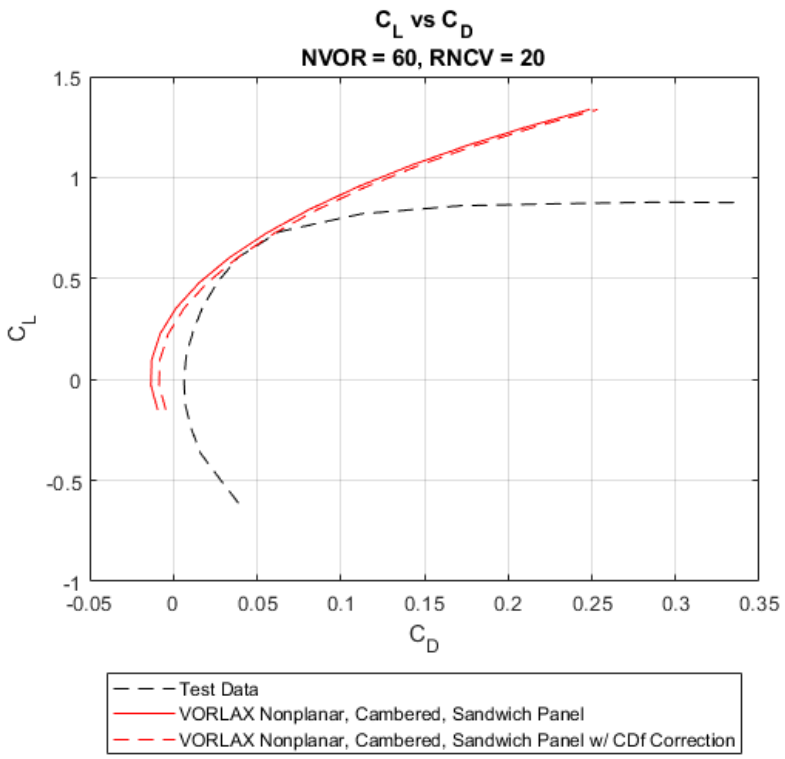


Figure 44b: Drag Polar for Nonplanar, Cambered, Sandwich Panel

Figures 43 and 44 show both the planar and nonplanar representations of a cambered panel of finite thickness. As shown in the earlier grid evaluations, the lift coefficients given by the nonplanar model are higher than those given by the planar models, however they both provide the expected linear increase in lift.

The drag polar plots vary drastically, however. For the planar case, the drag measurements are incorrect; they have the wrong sign. For the nonplanar case, the drag coefficients are pointing in the correct direction, however they relate to the measured drag figures only tangentially, thereby reaffirming the belief that using the sandwich panels for induced drag measurement is unreliable.

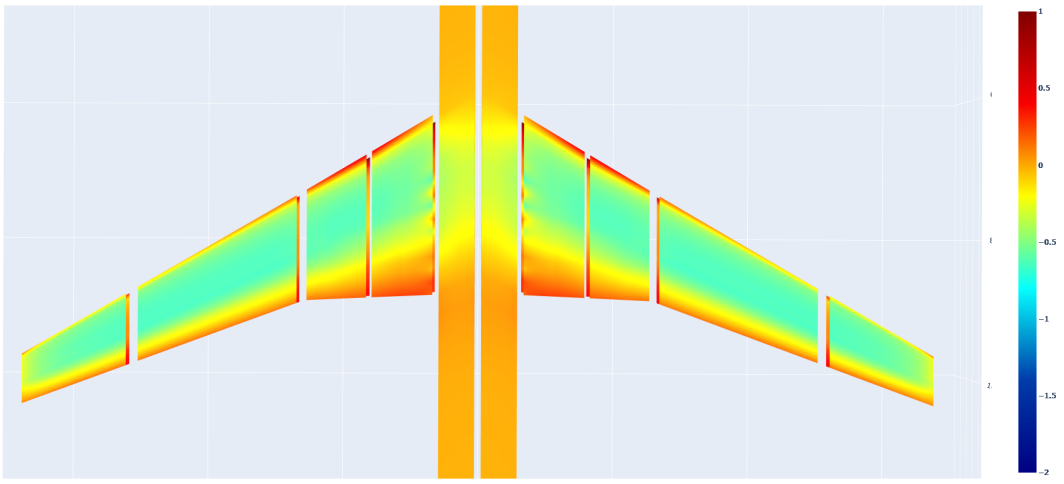


Figure 45: Pressure Contour for Cambered Sandwich Panels in Aircraft Configuration

The most accurate representation of an airfoil shape within the *VORLAX* program comes from the superposition of both thickness and camber perturbations. This mode is particularly useful in that it allows the user to model a wing with pressure loading remarkably well, giving massive insights to the local flow behavior on the wing surface. Like earlier modes, this mode will also allow the user to represent the wing as either planar or nonplanar, giving different results depending on the mode.

For an aerospace engineer working on an aircraft wing, this combination is the only one that allows for a truly accurate, workable pressure distribution. Recall that *VORLAX* takes station inputs in the form of (x/c) chordwise stations with (y/c) displacement scaling percentages. Thus, it is common to break a wing into five to six sections controlled by four to five unique control stations, respectively. Using these stations, an Excel sheet may be constructed to write the necessary *VORLAX* inputs very procedurally. Thus, dimension agreement can be enforced via cell calculations

and it is possible to build "rough estimate" visualizers based on the macroscopic dimensions.

Figure 46: Typical Example of VORLAX Interface via Spreadsheet

Some may also find it convenient to have quick access to numerous wing shapes. Using MATLAB and its curve-fitting toolbox, files containing ordinate and displacement coordinates can be automatically generated based on a text file coordinate input. This is advantageous as it allows the user to standardize the thickness and camber profiles for input construction. Recall that the camber and thickness effects are combined with superposition, and as such it is massively convenient to work with the same sets of (x/c) coordinates in the *VORLAX* input construction sheet. There will be some negligible error introduced by a MATLAB spline fit and evaluation,

however the tools are well-tuned for the two-dimensional geometries. Furthermore, the user is presented with a plot showing the input points, the smoothed spline fit, and the calculated points, which offers the chance to reject the fit. MATLAB may introduce some small errors to the fit (small at a magnitude where the difference is largely indistinguishable in the plot previews), however *VORLAX* itself also completes some interpolation between the points, almost certainly introducing small errors of the same order of magnitude as the MATLAB fit.

5.6 Pressure Coefficient Distribution

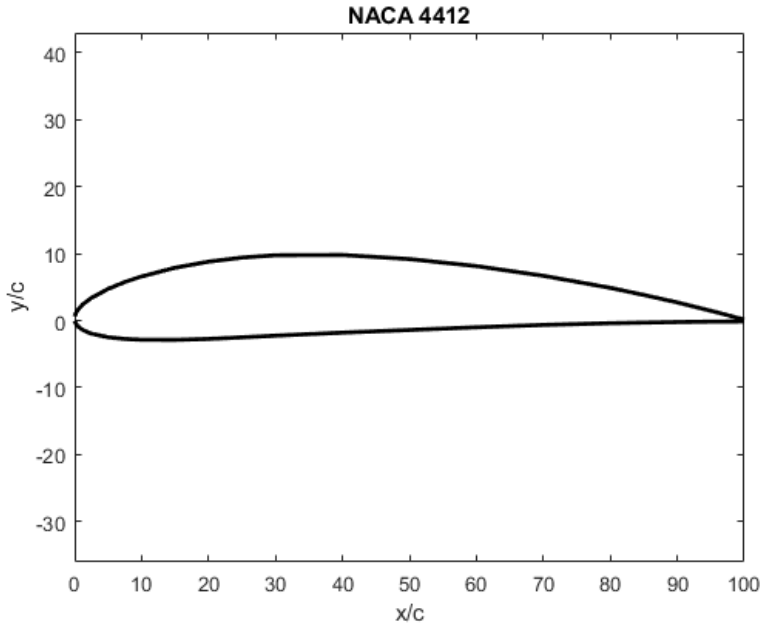


Figure 47: NACA 4412 Airfoil Profile

When using *VORLAX* to compute pressures on wings, it is important to obtain accurate figures that provide reasonable correlation with the real-world aerodynamic properties of the wing. To demonstrate this correlation, *VORLAX* was run for the

case of an incompressible NACA 4412 wing to compare with the quantitative pressure data presented in [22]. This study presented high-quality pressure distributions for the wing in the Langley variable-density wind tunnel, offering the opportunity to see the specific nature of the *VORLAX* pressure correlations.

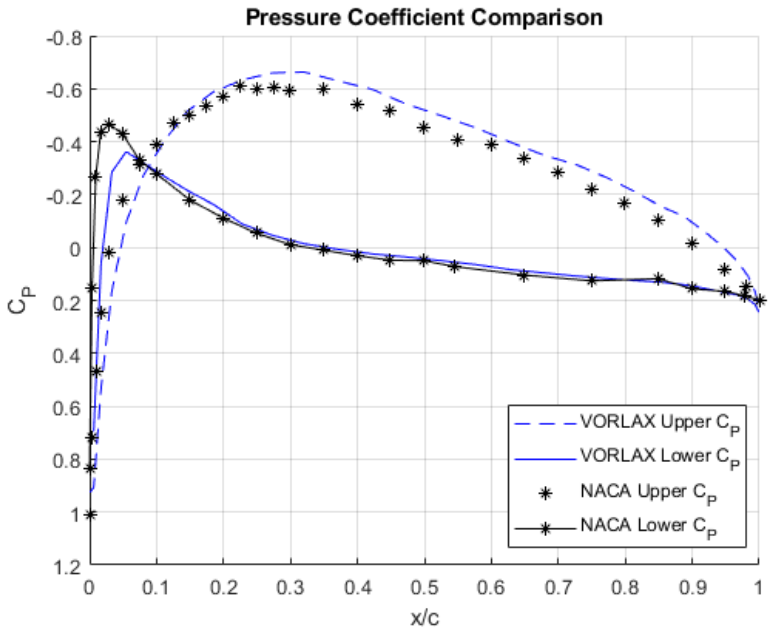


Figure 48: Pressure Coefficient Distribution about NACA 4412 Wing of Finite Span

Figure 47 shows the shape of the NACA 4412 profile as input into *VORLAX*. Figure 48 shows the results of the pressure distribution comparison. While the results are not identical, they are remarkably close at the centerline of the wing. The differences arise primarily from the lack of viscous effects (and therefore the lack of a boundary layer), so in the context of the Vortex-Lattice Method, the correlation is quite good. The magnitudes of the pressure coefficients are comparable and occur in approximately the same locations, thereby providing excellent insight into the nature of the flow behavior about the finite wing.

5.7 VORLAX Fusiform Bodies

VORLAX includes techniques to model cylindrical shapes using a fusiform body generation, which is particularly useful for modeling cylindrical aircraft body shapes as well as for modeling engine nacelles – both of which may generate interference to the airflow over an aircraft. The method does not require much modification to the input file other than the inclusion of polar coordinate pairs, however it allows the user to generate a very reasonable fuselage shape, seen in Figure 49.

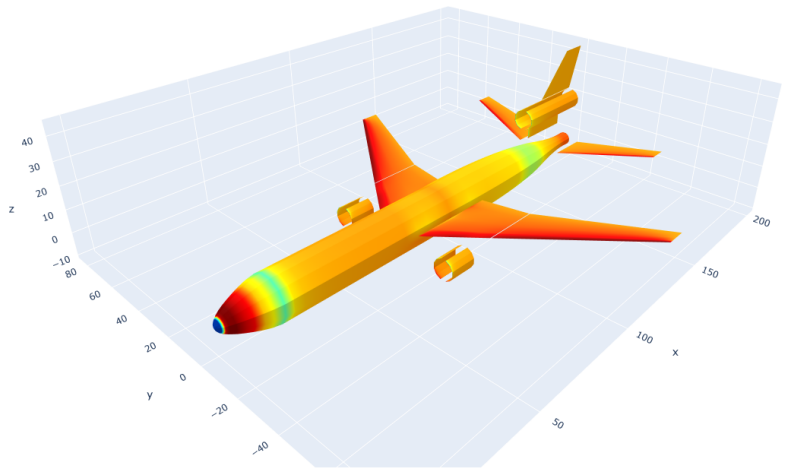


Figure 49: VORLAX Fusiform Model of McDonnell-Douglas MD-11

5.8 VORLAX Wake Survey

The *VORLAX* program offers wake survey functionality by the addition of a couple of simple flags near the footer of the input file. Wake surveying can be very useful due to its ability to present wash components due to a body in passing airflow. Some may wish to use the wake survey to resolve blunt body drag, for example. Another technique useful via the wake survey option is for engine design and placement. In the case of a propeller-driven aircraft, the user may utilize the wake survey feature for a plane directly in front of the nose of the aircraft in order to align the propellors in a manner that preserves the integrity of the flow distribution over the main wing and body.

The functionality of the wake survey tool is familiarly simple, following the common theme of the vortex-lattice program. The user specifies the number of desired planes to take a "cut" of the airflow in order to survey the wake and its velocity components within that plane. This $y - z$ plane is placed at an x-coordinate station in a user-defined location in the proximity of the aircraft and will return an output featuring the cartesian components and the velocity components at those locations. With this data, the user may easily utilize MATLAB, Excel, or Python to obtain contours detailing this flow.

Taking it a step further, with nothing other than simple knowledge of basic aerodynamic principles and numerical methods, the user may employ a finite-difference method to calculate the vorticity at each point. For the following example, this was done using second-order central difference schemes

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{5.2}$$

for the inner points of the grid and using first-order forward and backwards differences

at the edges at the grid.

$$f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{5.3}$$

$$f'(x) \approx \frac{f(x) - f(x-h)}{h} \tag{5.4}$$

While the quantities at the edge of the plane domain contain slightly larger errors than those in the inner portion, the wake survey should be sized in such a way that the edges are considered "far field", and do not pertain to the vortex behavior in the immediate proximity of the body.

Recall that vorticity is given via

$$\vec{\omega} \equiv \nabla \times \vec{u} \tag{5.5}$$

and because *VORLAX* gives the exact velocities over the grid, the finite-difference equations are perfectly adequate to approximate the derivatives of the velocities in space necessary to obtain the vorticity. Furthermore, because this is a matter of a 2-D planar "cut" in space, the only vorticity of interest is that in the x-direction, given as

$$\omega_x \equiv \frac{\partial w}{\partial y} - \frac{\partial v}{\partial z} \tag{5.6}$$

Thus, the user may determine the vorticity concentrations within the cut.

As an example, the wake was analyzed for an $AR = 2$ wing at a Mach number of 0.3 and 5 degrees of pitch. The grid used to compute the wake was fine, having 75 stations in the spanwise direction and 25 stations in the vertical direction. The wake was analyzed in two locations, one directly in front of the aircraft, and another slightly behind the rearmost trailing edge coordinate of the wing. As expected, the plane just aft of the trailing edge had much more spatial variation than the flow in the crossplane just ahead of the nose, however, there was still some interesting behavior.

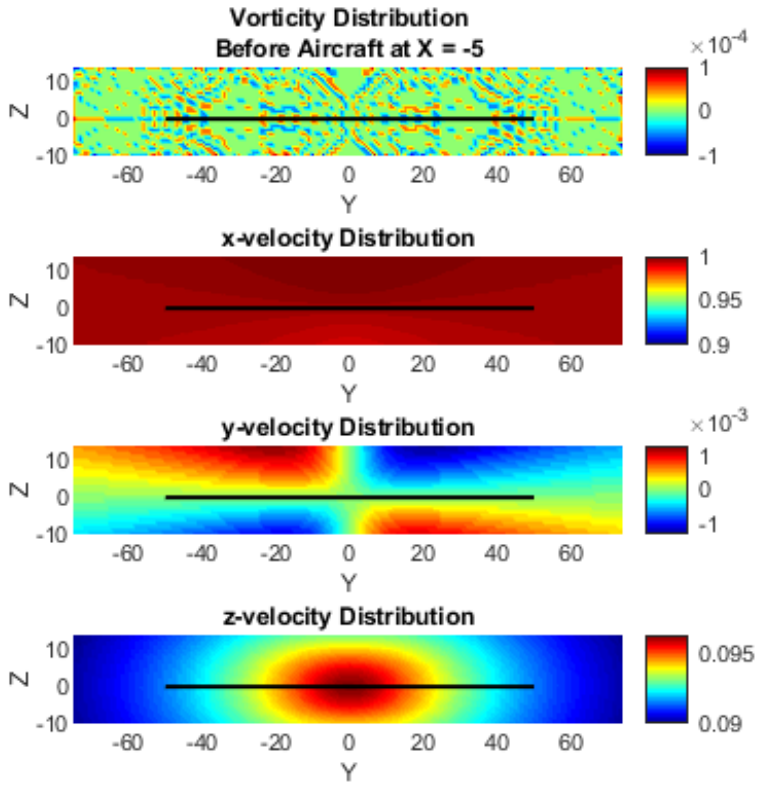


Figure 50: Freestream Wake Survey of Finite Wing

The vorticity distribution shows nothing other than extremely small fluctuations. While small and essentially negligible, the values are nonzero and are split between positive and negative values of vorticity. Recall that the sign notation for vorticity follows the right hand rule, so those values greater than zero indicate a vorticity concentration that would act counter-clockwise in this view, while those less than zero indicate a clockwise direction. The black line visible in the contour plot represents the location of the wing panel (this was a planar configuration) and was included for reference.

The incoming freestream x-component of velocity (the primary direction of the freestream flow) is largely constant. One takeaway from this plot is that the velocity

in the region above the wing is slightly higher in magnitude than that below the wing. This implies that there are some other velocities that exist to take away from the energy in the x-direction velocity. A portion of this change comes from the induced velocities associated with the upwash component of the flow.

The y-velocity profile shows that there are incredibly small velocities occurring in the spanwise direction, even prior to the airflow reaching the aircraft. This corresponds with the change to the x-velocities. On the lower portion of the plot, the velocities indicate that the small flows are moving away from the aircraft center plane on the bottom portion while they move towards the center plane on the upper surface. This is consistent with the contours for vorticity, in which there were small yet visible vortices visible near the wingtips.

Finally, the contours for the z-velocity components are some of the most interesting. Although this "slice" of the flow occurs prior to the wing passing through the location in space, there is a considerable magnitude of air with an induced vertical velocity. Namely, there is an upwash component equal to 9% of the x-velocity visible in the region just before the aircraft. Thus, the true direction of the flow normal to the wing does not occur head-on, but rather it occurs with a small upward component. This information, while interesting to look at, also serves a practical purpose in aircraft design applications. By understanding the direction of the airflow that the aircraft "sees", it becomes easier to design propellor systems and even turbofan systems optimally. For propellors, this means that the effects due to the propellors on the over-wing airflow can be minimized and accounted for in design. As for turbofan applications, this becomes useful when arranging the diffuser, in which it is desirable to have a geometry that maintains shock-free flow during engine operation.

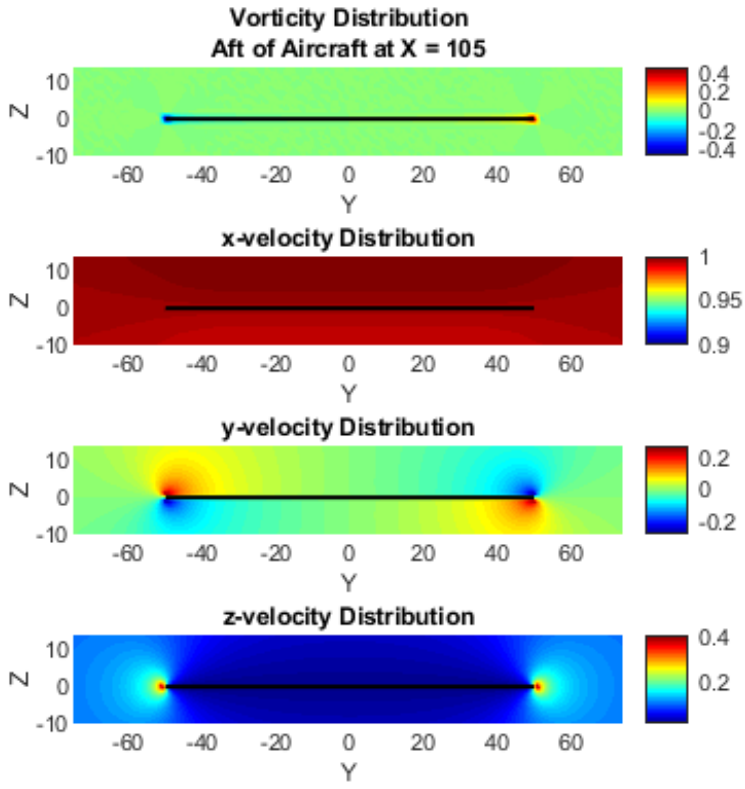


Figure 51: Trailing Wake Survey of Finite Wing

For the survey occurring in the region aft of the wing, the contours show airflow conditions that are considerably different than those in the region before the wing. The vortices are now much more concentrated, with large vorticity magnitudes occurring at the wingtips. These are the well-understood vortices that lead to increases in induced drag, and they are largely due to the large pressure imbalance between the upper and lower surface of the wingtips. Much like the region before the aircraft, the x-velocity (out of the page) is largely uniform, but with small differences in the regions above and below the wing. Likewise, the y-velocity shows the same behavior as in Figure 50, however there are now much larger magnitudes that are concentrated at the wingtips, rather than small magnitudes occurring in the entire proximity near

the wing. It should be noted that these y-direction velocities on the surface of the wing are largely the reason that two-dimensional approximations of airflow on a wing are hugely inaccurate. The velocities in this model show that the magnitudes of the spanwise velocity components are of the order of 20% the freestream flow velocity, which is considerably large.

Finally, the components of velocity in the z-direction have shifted towards the wingtips, becoming much more concentrated. This is a large difference from the region causing upwash before the aircraft, as the z-components of the velocities are now acting in a much tighter region, but are doing so at a magnitude four times those before the aircraft. In both cases, the z-velocities are entirely positive. This leads to a decreased effective angle of attack for the wing of the aircraft, again agreeing with the commonly understood behavior of induced drag.

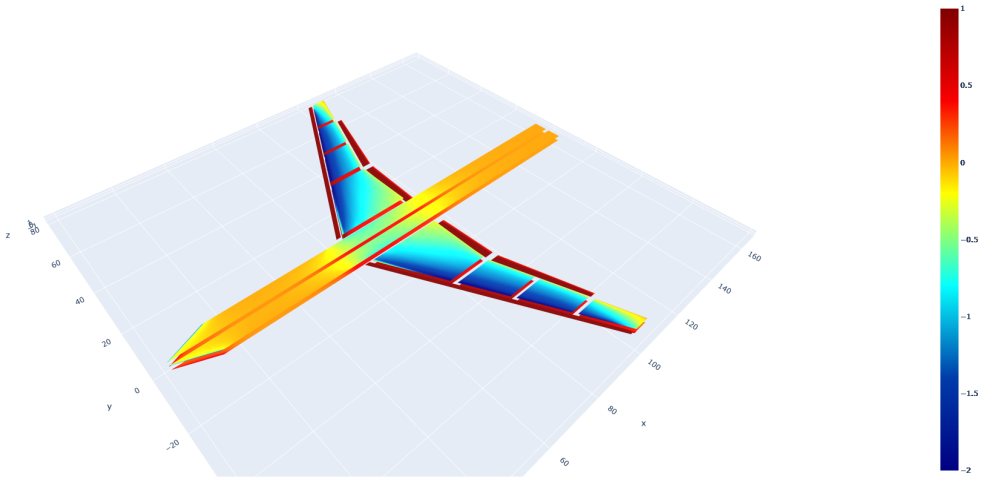


Figure 52: VORLAX Model of Aircraft Flap Configuration [23]

As another demonstration for the wake surveying feature of *VORLAX*, a wake survey was completed for a model of an entire airframe including flaps, courtesy of

another one of Professor Takahashi's graduate students [23]. This model involves a flat representation of a fuselage, complete with wings including twist, thickness, and camber. The noteworthy feature of this model is that includes flat panel high-lift devices at the leading- and trailing-edge of the main wing of the aircraft. This was drawn using VORLAXPlot.py, a tool developed as a part of this thesis and detailed further in Chapter 6.

This aircraft was modeled at Mach 0.2 with an aggressive angle of attack of 12 degrees, representing an aircraft landing configuration for a narrow body aircraft comparable to a Boeing 757 in size. Due to the drastic configuration changes, it was expected that the wake caused by the body would be considerably different than that of the simple wing.

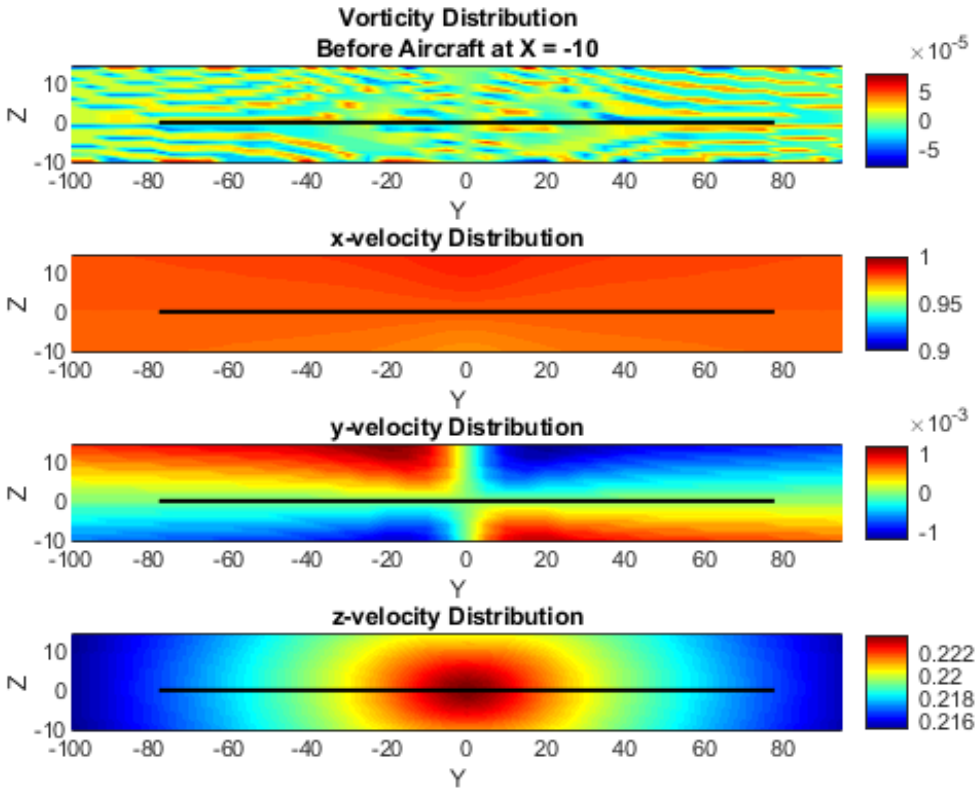


Figure 53: Freestream Wake Survey for Aircraft with High-Lift Devices

The behavior of the wake before the aircraft is largely similar to that of the single wing in a flow. The vorticity contour shows very small vortices that are largely symmetric about the wing. The x-velocity again shows a nearly uniform behavior; however, it is at a slightly decreased magnitude compared to the single wing. The spanwise flow is nearly identical to that of the single wing. Finally, the z-velocity features a similar contour pattern, but the magnitude is significantly higher than that of the single wing. This is believable because this model was of a large aircraft at a high angle of attack at a lesser speed, and so there would be larger velocity fluctuations in the vertical direction because of the attached-flow assumption.

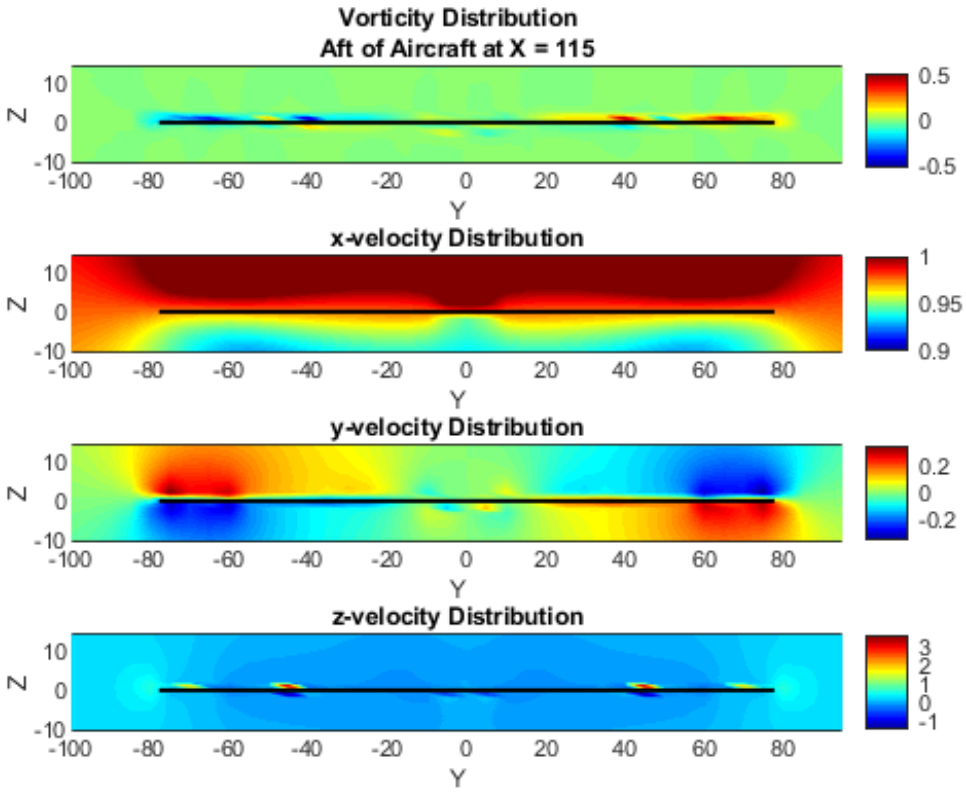


Figure 54: Trailing Wake Survey for Aircraft with High-Lift Devices

Figure 54 shows the results for the wake aft of the aircraft. It is noticeable that the vorticity for this configuration occurs at a slightly higher concentration at more specific locations relative to the simple "wing in a flow" configuration. The locations of this vorticity concentration occur at expected regions, namely the wingtips and the edges of the trailing-edge flaps. Recall that the vorticity is understood to arise from pressure imbalance with no physical device existing to maintain that imbalance, and thus it makes sense that the vorticity concentration is in regions supporting high levels of lift near edges of surfaces.

The x-velocity components are significantly different than those in the prior example. While the simple wing only showed the velocity decreasing to 97-98% of

its freestream magnitude in the region under the wing, this model with flaps shows a decrease to 90-97% in the regions below, much slower than the other case. For an aircraft in a landing configuration, the wing is making much more lift, which leads to a much larger component of flow acting downwards in the region below the aircraft. With this, there is also a large accumulation of pressure under the wing, further causing a decrease in the x-component of the velocity.

The y-velocity contours share similar information as those in the single wing configuration. In the flow aft of the wing, there is spanwise velocity induced towards the center plane of the aircraft above the wing, while the velocity points away from the center plane of the aircraft below the wing. While the direction and magnitudes of these velocities remains the same for the aircraft in the landing configuration, the regions where these magnitudes occur were restricted largely to the wingtips, with considerably smaller spanwise components moving in towards the center plane. However, in the landing configurations, the spanwise components stay at higher magnitudes covering a significantly larger portion of the wing. These components still become zero towards the center, but this happens over a much smaller spanwise distance.

Finally, the z-velocities show that there are incredibly large upwards components occurring aft of the aircraft above the main wing near the gap between trailing edge flaps. Likewise, there is a slightly smaller downward component of flow below the wing in the same region. This is considerably different than the behavior for the single wing case, as there was very little z-velocity in the wake behind the wing. In this case, the components at the wingtips are smaller than those for the lone wing, and there are large but concentrated components near the flap gaps under the primary wing.

5.9 Stability and Control via VORLAX

VORLAX is able to provide the information necessary to compute stability and control parameters of an aircraft. The panel method is very versatile for representing control surfaces. Ailerons, elevators, and rudders may be modeled using a standard flat panel, and its deflection is represented using the panel twist parameter in its input card. Thus, it is possible to fully automate a basic stability and control spreadsheet for use in early design stages.

To automate the process, a scripting language such as VBA is used to run through five *VORLAX* input models, including a neutral configuration, a model with one degree of sideslip, and then one model for each of the control surfaces deflected to their maximum angle. By doing so, one may recover all major stability derivatives, and with nothing more than basic cell arithmetic, can present stability and controllability outcomes to a reasonable degree of certainty. Of these include the crosswind performance, Dutch-Roll frequency, stick-fixed frequency, control speeds, and many more. It is also possible to generate nearly every type of stability and control plot, some examples of which are seen in Figure 55, as well as an interface for the performance calculations in Figure 56.

Prior to the update to *VORLAX*, the stability and control analysis was incredibly slow. Each of the five models is run at 14 angles of attack and three Mach numbers, and as a result the runtimes were nearly 10 minutes per run, which was necessary for each alteration to the aircraft, no matter the size. With the *VORLAX* updates, this time was reduced drastically, now taking only a matter of seconds per run. This greatly streamlines the stability and control analysis of an aircraft using *VORLAX*.

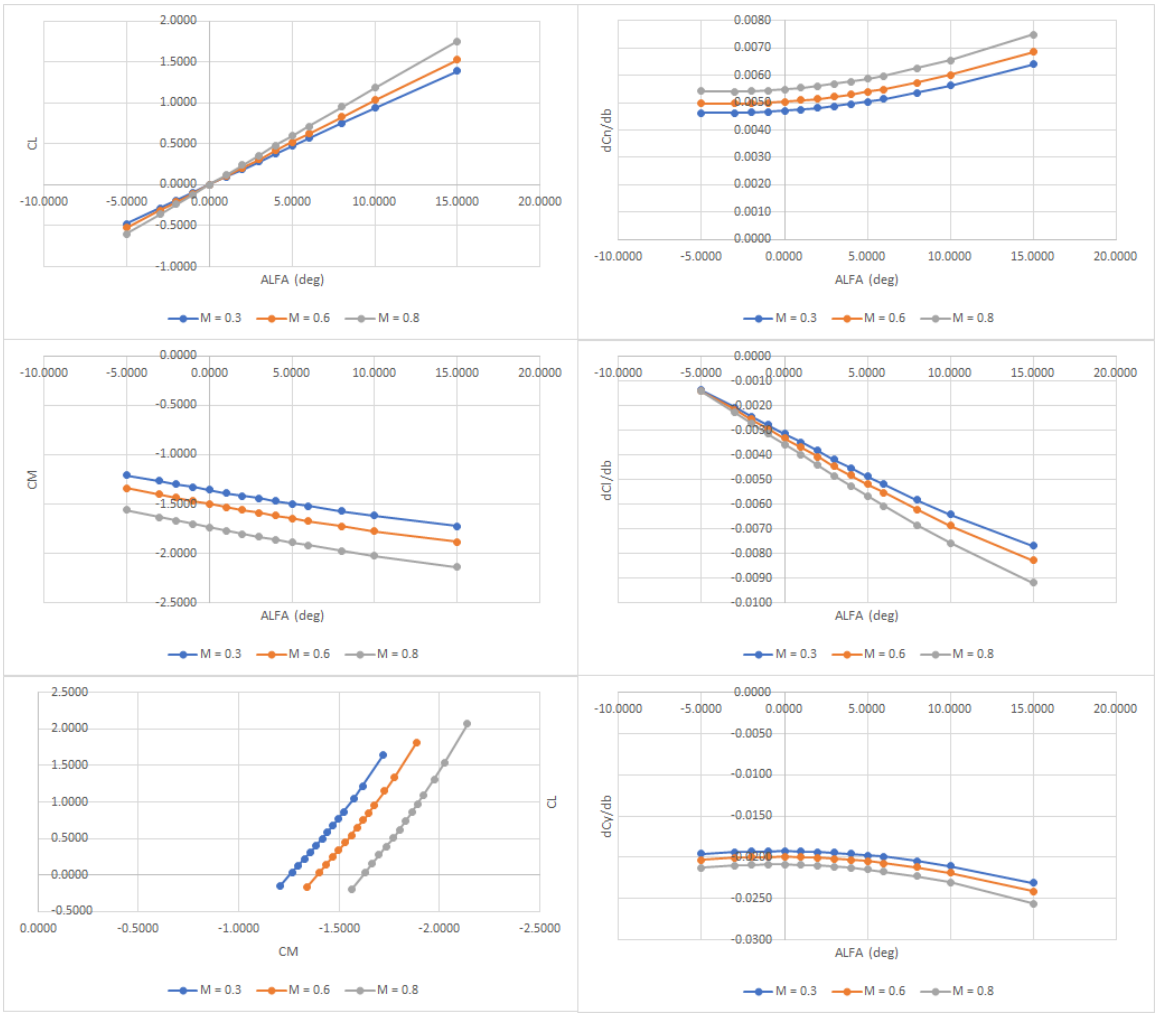
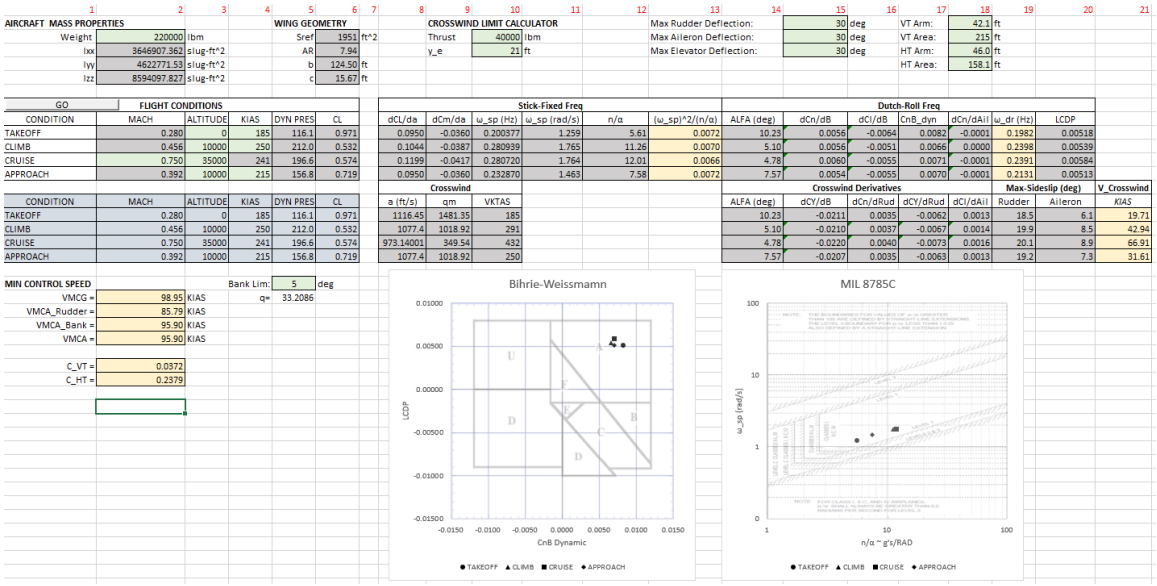


Figure 55: Several Stability and Control Plot Examples



AIRCRAFT MASS PROPERTIES

Weight	220000	lbm
Ixx	3646907.362	slug-ft^2
Iyy	4622771.53	slug-ft^2
Izz	8594097.827	slug-ft^2

WING GEOMETRY

Sref	1951	ft^2
AR	7.94	
b	124.50	ft
c	15.67	ft

CROSSWIND LIMIT CALCULATOR

Thrust	40000	lbm
y_e	21	ft

Max Rudder Deflection:	30	deg
Max Aileron Deflection:	30	deg
Max Elevator Deflection:	30	deg

VT Arm:	42.1	ft
VT Area:	215	ft
HT Arm:	46.0	ft
HT Area:	158.1	ft

GO

FLIGHT CONDITIONS

CONDITION	MACH	ALTITUDE	KIAS	DYN PRES	CL
TAKEOFF	0.280	0	185	116.1	0.971
CLIMB	0.456	10000	250	212.0	0.532
CRUISE	0.750	35000	241	196.6	0.574
APPROACH	0.392	10000	215	156.8	0.719

Stick-Fixed Freq

dCL/da	dCm/da	ω_sp (Hz)	ω_sp (rad/s)	n/α	(ω_sp)^2/(n/α)
0.0950	-0.0360	0.200377	1.259	5.61	0.0072
0.1044	-0.0387	0.280939	1.765	11.26	0.0070
0.1199	-0.0417	0.280720	1.764	12.01	0.0066
0.0950	-0.0360	0.232870	1.463	7.58	0.0072

Dutch-Roll Freq

ALFA (deg)	dCn/dB	dCl/dB	CnB_dyn	dCn/dAil	ω_dr (Hz)	LCDP
10.23	0.0056	-0.0064	0.0082	-0.0001	0.1982	0.00518
5.10	0.0056	-0.0051	0.0066	0.0000	0.2398	0.00539
4.78	0.0060	-0.0055	0.0071	-0.0001	0.2391	0.00584
7.57	0.0054	-0.0055	0.0070	-0.0001	0.2131	0.00513

Crosswind

CONDITION	MACH	ALTITUDE	KIAS	DYN PRES	CL	a (ft/s)	qm	VKTAS
TAKEOFF	0.280	0	185	116.1	0.971	1116.45	1481.35	185
CLIMB	0.456	10000	250	212.0	0.532	1077.4	1018.92	291
CRUISE	0.750	35000	241	196.6	0.574	973.14001	349.54	432
APPROACH	0.392	10000	215	156.8	0.719	1077.4	1018.92	250

Crosswind Derivatives

ALFA (deg)	dCY/dB	dCn/dRud	dCY/dRud	dCl/dAil	Max-Sideslip (deg) Rudder	Aileron	V_Crosswind *KIAS*
10.23	-0.0211	0.0035	-0.0062	0.0013	18.5	6.1	19.71
5.10	-0.0210	0.0037	-0.0067	0.0014	19.9	8.5	42.94
4.78	-0.0220	0.0040	-0.0073	0.0016	20.1	8.9	66.91
7.57	-0.0207	0.0035	-0.0063	0.0013	19.2	7.3	31.61

MIN CONTROL SPEED

Bank Lim: 5 deg

q= 33.2086

VMCG =	98.95	KIAS
VMCA_Rudder =	85.79	KIAS
VMCA_Bank =	95.90	KIAS
VMCA =	95.90	KIAS

C_VT =	0.0372
C_HT =	0.2379

Figure 56: VORLAX Stability and Control Performance Spreadsheet

CHAPTER 6

VORLAX VISUALIZER DEVELOPMENT

One aspect of *VORLAX* that has been mentioned repeatedly in this document is the fact that not only are the preprocessing and runtimes fast and efficient, but so are the post-processing tasks. The utilization chapter has shown the various quantities that one may obtain from a *VORLAX* run, but as with most CFD programs, sometimes the user is simply looking for a pretty visual that they may paste onto their next PDR presentation. Furthermore, with the "all in one" nature of *VORLAX*, having good post-processing tools provides the user the capability of debugging their input parameters and rectifying any geometry errors in their input.

Professor Takahashi has offered a visualizer for both his professional usage and his students' educational purposes. This visualizer is written in VBA and offers a wireframe representation of the *VORLAX* input, and it does so in a manner that allows the user to see the overall view of the *VORLAX* model before running it. This is particularly nice because it is useful for strictly preprocessing, with the tradeoff being that it runs very quickly. With this visualizer, the user is able to see the general proportions of the model, as well as overlays showing the twist and camber of each panel. It is a great tool that has seen considerable use, especially when linked to trade study software such as ModelCenter [24] [25] [26].

While this visualizer remains very useful, there are some addressable drawbacks that made it worthwhile to develop a new visualizer. The first requirement is that the visualizer utilizes only the input file, which inherently limits it from being able to provide too much information about a given model. There is also the fact that

the model is bound by the limitations imposed by Excel. Thus, there are only three available views: one planform, one side profile, and one isometric. These views are limited to a user-defined scaling factor within the spreadsheet. Finally, the proportions of the model are susceptible to the scaling handled by Microsoft, which may vary between versions of the Excel software. There have been cases where models look incorrect on the Excel-based visualizer, despite being fine. This is particularly noticeable on high-resolution monitors.

The few drawbacks on the old visualizer inspired the creation of a new visualizer that would operate in a complimentary fashion to the old visualizer. The goal of the new visualizer was that it would offer more modern visual functionality, including zooming, panning, and rotating of the model. Fortunately, the *VORLAX* geometric generation system involves standard cartesian coordinates, making this a trivial task. Another feature desired within the new visualizer was the ability to integrate pressure data to analyze isobars over the aerodynamic surfaces. Having these goals in mind, the new tool was created.

VORLAXPlot.py is a python-based visualization utility that is compatible with the *VORLAX* log output files and allows the user to quickly generate a detailed model using a simple command line interface. Python was chosen due to its cross-platform capabilities, allowing the user to generate models from anywhere, even if they use an Apple or Linux OS on their laptop with a Windows desktop. The tool makes use of common, open-source Python libraries, namely Plotly and Numpy. Plotly is a fantastic, vetted utility that may replicate the behavior of a proprietary software such as MATLAB without the same licensing requirements. With Plotly, the user is able to pan, zoom, and rotate the model all within a standard web browser in an HTML file format, which proves incredibly useful to a standard user. The only complication with

the program is that the user must have Python 3 installed on their system, however the software is completely free and trivial to install, either with pip or conda used to obtain the necessary libraries.

The python tool will plot the model using the X, Y, and Z coordinates returned by *VORLAX*. The code is structured in a such a manner that it remains completely robust regarding the grid spacings, panel sizes, panel permeabilities, and any other variable factors. Each panel is treated as an object, having its sizing and grid parameters stored within one object variable, and then the parameters are later called to parse through the pressures. The only assumption in the code is that if there is a fusiform panel, it is the first listed in the *VORLAX* input, which is standard practice as it is.

The three-dimensional models include isobar data pulled from the DCP values for each X, Y, and Z point. This allows the user to easily see the behaviors of the pressure contours over the entire body, no matter the shape. Due to the generalized nature of the visualizer, if the user chooses to model only a wing (having no body), it works perfectly, maintaining the same functionality as the entire airframe models. By varying the NPP value, the user may also alter their model such that the visualizer will draw the actual displacements of the camber and thickness effects. Thus, with minimal effort, the user may obtain remarkably accurate model visualizations. The capability to support fusiform-style configurations is new within a *VORLAX* visualizer and has been useful for detailing the actual operations happening when a user specifies a fusiform body. Within the visualizer code, there is an additional function that resolves the gaps between the fusiform panels in order to make the cylinder continuous. This does not introduce inaccuracy into the model, as it still bases the numerical values from data at each of the discrete points. This simply introduces extra shading to ensure that the body appears as the user intended.

VORLAX allows the user to specify panels that are not symmetric about the center plane of the aircraft. As a result, the program must have the ability to adjust the contour and mirroring settings as necessary. This was accomplished by using the built-in *VORLAX* "IQUANT" flag, which specifies the symmetric properties about the X-Z plane. Without this flag, the program would incorrectly layer "ghost" mirrored panels on top of one another, giving incorrect visuals.

This utility has been utilized in the ongoing aerospace engineering capstone project, as well as for the research projects of Professor Takahashi's other graduate students. From this deployment, it has been apparent that the visualizer is not only useful for creating visually appealing models, but it serves a massive purpose as a debugging tool. Because the old visualizer was done in Excel using wireframe drawings, there was a limit to the amount of information one could obtain from them. With the new tool, common errors such as panel disagreements, scaling errors, and irregular spacing become much easier to identify.

As a final perk, the new visualizer allows the user to save the visuals quite easily, and sharing is also incredibly easy because the visuals can be rendered on any computer with a web browser. This has made it easier to debug models both for students and for colleagues. It has been immensely easier to visualize and share models to others working with *VORLAX* because everything can be handled via individual files, rather than by re-running the visualizer numerous times when working with the Excel tool.

CHAPTER 7

CONCLUSION

This work has served as a detailed overview regarding the usage, optimization, and capabilities of the generalized Vortex-Lattice Method by using the program *VORLAX*. Through numerous examples, it has been shown that while the method is incredibly mature and has fallen from use, it is an incredibly capable and viable method for resolving the pressure qualities of airflow over a surface. While the method is bound by a number of assumptions, it is understood that the performance within those bounds is both robust and reliable, should the user understand the limitations at the program.

Numerically, the program operates quickly with a reasonable number of grid points. While the 32-bit limitation of the program causes memory constraints, the results have shown that the models are largely insensitive to grid densities. The program is quick to generate and solve the linear systems required to resolve the flow characteristics and has been shown to be adaptable in a manner that allows rapid runs of the program back-to-back.

The pre-processing has been shown to be fast and consistent – properties which make the program simple to automate. Likewise, it has been shown that the outputs are equally as consistent, allowing the user to utilize the same set of custom tools repeatedly for a number of geometries. Because *VORLAX* operates quite simply, it is safe to say that its usage in education and early design cases is unrivaled. For a very low cost, the user can obtain results that both build intuition and showcase probable behavior for a number of aviation designs of all sizes.

While the vortex-lattice method is fantastic and under-appreciated, nowhere in this study has it been touted as the end-all be-all solution to all aerodynamics solutions. It has been shown to be an immensely powerful tool that is robust to the point of validating early design considerations, performing sizing experiments on control surfaces, and examining airfoil geometry behavior. There is and will always be a place for high-fidelity CFD programs as well as experimental testing, but the vortex-lattice method is incredibly competitive and deserves to see modern usage solely due to its favorable ratio between running cost and performance.

REFERENCES

[1] Luis R. Miranda, Robert D. Elliot, and William M. Baker. *A Generalized Vortex Lattice Method for Subsonic and Supersonic Flow Applications*. Tech. rep. 1977.

[2] IBM. *No Title*. URL: https://www.ibm.com/ibm/history/exhibits/mainframe/mainframe_PP3155.html.

[3] Eben Upton. *Raspberry Pi Zero: the $5 computer*. 2015.

[4] Ifixit. *iPhone 12 and 12 Pro Teardown*. 2020. URL: https://www.ifixit.com/Teardown/iPhone+12+and+12+Pro+Teardown/137669 (visited on 03/27/2021).

[5] Andrei Frumusanu. *The 2020 Mac Mini Unleashed: Putting Apple Silicon M1 To The Test*. 2020. URL: https://www.anandtech.com/show/16252/mac-mini-apple-m1-tested.

[6] Lenovo. *No Title*. URL: https://www.lenovo.com/us/en/laptops/thinkpad/thinkpad-p/ThinkPad-P1-Gen-3/p/22WS2P1P1N3.

[7] Tomas Melin. "A vortex lattice MATLAB implementation for linear aerodynamic wing applications". MA thesis. Royal Institute of Technology, 2000.

[8] Timothy T. Takahashi. *Aircraft Performance and Sizing, Volume I*. Momentum Press, 2017.

[9] Tyler J Souders and Timothy T Takahashi. "VORLAX 2020: Benchmarking Examples of a Modernized Potential Flow Solver". In: *AIAA AVIATION*. 2021.

[10] Ira H. Abbott and Albert E. Von Doenhoff. *Theory of wing sections: including a summary of airfoil data*. Courier Corporation, 2012.

[11] T. P. Kalman, J. P. Giesing, and W. P. Rodden. "Reply by Authors to G. J. Hancock". In: *Journal of Aircraft* 8.8 (1971), pp. 681–682.

[12] Marcus Herrmann. *Course Lecture*. Tempe, 2020.

[13] Richard C. Feagin and William D. Morrison Jr. *Delta Method, an Empirical Drag Buildup Technique*. Tech. rep. Burbank, CA: Lockheed-California Company, 1978.

[14] G.L. Martin. *Modifications to the WDTVOR and VORTWD computer programs for converting input data between VORLAX and wave drag input formats*. Tech. rep. 1978.

[15] Dietrich Kuchemann. *The Aerodynamic Design of Aircraft*. American Institute of Aeronautics and Astronautics, 2012.

[16] John D. Anderson. *Modern Compreesible Flow*. Tata McGraw-Hill Education, 2003.

[17] Edward C. Lan. "A quasi-vortex-lattice method in thin wing theory". In: *Journal of Aircraft* 11.9 (1974), pp. 518–527.

[18] Everett W Purcell. "The vector method of solving simultaneous linear equations". In: *Journal of Mathematics and Physics* 31.1-4 (1953), pp. 180–183.

[19] Henk A. Van der Vorst. *Iterative Krylov methods for large linear systems*. Cambridge University Press, 2003.

[20] Fred A. Demele and Fred B. Sutton. *The Effects of Increasing the Leading-Edge Radius and Adding Forward Camber on the Aerodynamic Characteristics of a Wing with 35 degrees of Sweepback*. Tech. rep. Washington: National Advisory Committee for Aeronautics, 1951.

[21] James Jensen and Timothy T. Takahashi. "Wing Design Challenges Explained: A Study of the Finite Wing Effects of Camber, Thickness, and Twist". In: 2015.

[22] Robert M Pinkerton. *Calculated and Measured Pressure Distributions Over the Midspan Section of the N.A.C.A. 4412 Airfoil*. Tech. rep. NATIONAL ADVISORY COMMITTEE FOR AERONAUTICS, 1936.

[23] Gabino Martinez Rodriguez. "A Study and Design of Multi-Element High Lift Systems for Transport Aircraft". MA thesis. Arizona State University, 2021.

[24] Timothy T. Takahashi. *VORLAX Model Visualizer*.

[25] Phoenix Integration. *ModelCenter*. 2021.

[26] Timothy T. Takahashi. *Aircraft Performance and Sizing, Volume II*. Momentum Press, 2017.

APPENDIX A

EXAMPLE VORLAX INPUT FILES

A.1 Flat Panel with no Camber

```
FLAT PANEL WITH NO CAMBER
*00000000111111111122222222223333333333444444444455555555556666666666
*23456789012345678901234567890123456789012345678901234567890123456789012345678
*ISOLV   LAX      LAY      REXPAR HAG      FLOATX FLOATY
    ITRMAX
0.0      0.0      0.0      0.2      0.0      0.0      0.0      399.0
*NMACH MACH
1.0       0.21
*NALFA  ALPHA
14.0       −10 −8 −6 −4 −2 0 2 4 6 8 10 12 14 16
*LATRL  PSI      PITCHQ ROLLQ   YAWQ    VINF
0.0      0.0      0.0      0.0      0.0      1.0
*NPAN   SREF     CBAR     XBAR     ZBAR     WSPAN
1.0      1279.584 16.84    4.21     0.0      76.0
*−−−−−−−−−−−−−−−−−−−
* Horizontal Tail
*X1      Y1       Z1       CORD1
0.0      0.0      0.0      22.5
29.43    38.0     0.0      11.25
*NVOR   RNCV     SPC      PDL
060.0    20.0     1.0      0.0
*AINC1  ANINC2   ITS      NAP      IQUANT ISYNT    NPP
0.0      0.0      0.0      0.0      0.0      0.0      0.0
*−−−−−−−−−−−−−−−−−−−
* NXS    NYS      NZS
0.0      0.0      0.0
*
*
* END
```

```
FLAT PANEL WITH CAMBER
*000000001111111111222222222233333333334444444444555555555566666666666
*234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
*ISOLV   LAX      LAY      REXPAR HAG      FLOATX FLOATY
    ITRMAX
0.0      0.0      0.0      0.2      0.0      0.0      0.0      399.0
*NMACH MACH
1.0      0.21
*NALFA  ALPHA
14.0     −10 −8 −6 −4 −2 0 2 4 6 8 10 12 14 16
*LATRL  PSI      PITCHQ ROLLQ  YAWQ   VINF
0.0      0.0      0.0      0.0      0.0      1.0
*NPAN   SREF     CBAR     XBAR     ZBAR     WSPAN
1.0      1280.0   16.84    4.21     0.0      76.0
*−−−−−−−−−−−−−−−−−−−
* Horizontal Tail
*X1      Y1       Z1       CORD1
0.0      0.0      0.0      22.5
29.43    38.0     0.0      11.25
*NVOR   RNCV     SPC      PDL
060.0    20.0     1.00     0.0
*AINC1  ANINC2  ITS      NAP      IQUANT ISYNT   NPP
0.0      0.0      0.0      18.0      0.0      0.0      0.0
*−−−−−−−−−−−−−−−−−−−
*
* X/C
 0.0000
 1.2500
 2.5000
 5.0000
 7.5000
 10.0000
 15.0000
 20.0000
 25.0000
 30.0000
 40.0000
 50.0000
 60.0000
```

70.0000
80.0000
90.0000
95.0000
100.0000
* CAMBER ROOT
0.0000
0.4890
0.9580
1.8330
2.6250
3.3330
4.5000
5.3330
5.8330
6.0000
5.8780
5.5100
4.8980
4.0410
2.9390
1.5920
0.8270
0.0000
* CAMBER TIP
0.0000
0.4890
0.9580
1.8330
2.6250
3.3330
4.5000
5.3330
5.8330
6.0000
5.8780
5.5100
4.8980
4.0410
2.9390
1.5920
0.8270

0.0000
* NXS NYS NZS
0.0 0.0 0.0
* END

```
SANDWICH PANEL WITH NO CAMBER
*000000001111111111222222222233333333334444444444555555555566666666666
*234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
*ISOLV   LAX      LAY      REXPAR HAG      FLOATX FLOATY
    ITRMAX
0.0      0.0      0.0      0.2      0.0      0.0      0.0      399.0
*NMACH MACH
1.0       0.21
*NALFA  ALPHA
14.0      −10 −8 −6 −4 −2 0 2 4 6 8 10 12 14 16
*LATRL  PSI      PITCHQ ROLLQ  YAWQ   VINF
0.0      0.0      0.0      0.0      0.0      1.0
*NPAN   SREF     CBAR     XBAR     ZBAR     WSPAN
2.0      1279.584 16.84    4.21     0.0      76.0
*−−−−−−−−−−−−−−−−−−−
* Horizontal TailUPPER SANDWICH PANEL
*X1      Y1       Z1       CORD1
0.0      0.0      0.75     22.5
29.43    38.0     0.375    11.25
*NVOR   RNCV     SPC      PDL
060.0    20.0     1.0      0.0
*AINC1  ANINC2   ITS      NAP      IQUANT ISYNT    NPP
0.0      0.0      1.0      19.0     0.0      0.0      1.0
*−−−−−−−−−−−−−−−−−−−
* X/C
 0.0000
 0.5000
 0.7500
 1.2500
 2.5000
 5.0000
 7.5000
 10.0000
 15.0000
 20.0000
 30.0000
 40.0000
 50.0000
 60.0000
```

70.0000
80.0000
90.0000
95.0000
100.0000
* XLE1
0.6870
* CAMBER ROOT
0.0000
0.8000
0.9700
1.2300
1.6900
2.3300
2.8100
3.2000
3.8100
4.2700
4.8400
5.0000
4.6800
4.0200
3.1300
2.1000
1.0600
0.5400
0.0200
* XLE2
0.6870
* CAMBER TIP
0.0000
0.8000
0.9700
1.2300
1.6900
2.3300
2.8100
3.2000
3.8100
4.2700
4.8400
5.0000

4.6800
4.0200
3.1300
2.1000
1.0600
0.5400
0.0200
*−−−−−−−−−−−−−−−−−−
* Horizontal TailLOWER SANDWICH PANEL

*X1	Y1	Z1	CORD1			
0.0	0.0	−.75	22.5			
29.43	38.0	−.375	11.25			
*NVOR	RNCV	SPC	PDL			
060.0	20.0	1.0	0.0			
*AINC1	ANINC2	ITS	NAP	IQUANT	ISYNT	NPP
0.0	0.0	−1.0	19.0	0.0	0.0	1.0

*−−−−−−−−−−−−−−−−−−
* X/C
0.0000
0.5000
0.7500
1.2500
2.5000
5.0000
7.5000
10.0000
15.0000
20.0000
30.0000
40.0000
50.0000
60.0000
70.0000
80.0000
90.0000
95.0000
100.0000
* XLE1
0.6870
* CAMBER ROOT
0.0000
−0.8000

−0.9700
−1.2300
−1.6900
−2.3300
−2.8100
−3.2000
−3.8100
−4.2700
−4.8400
−5.0000
−4.6800
−4.0200
−3.1300
−2.1000
−1.0600
−0.5400
−0.0200
* XLE2
0.6870
* CAMBER TIP
0.0000
−0.8000
−0.9700
−1.2300
−1.6900
−2.3300
−2.8100
−3.2000
−3.8100
−4.2700
−4.8400
−5.0000
−4.6800
−4.0200
−3.1300
−2.1000
−1.0600
−0.5400
−0.0200
*−−−−−−−−−−−−−−−−−−
* NXS NYS NZS
0.0 0.0 0.0

∗ END

* Note: The configuration for a sandwich panel with camber is nearly identical, only with different values for the camber at each station.

```
MD−11 INPUT
∗00000000111111111122222222223333333333444444444455555555556666666666
∗23456789012345678901234567890123456789012345678901234567890123456789012345678
∗ISOLV   LAX      LAY      REXPAR HAG      FLOATX FLOATY
    ITRMAX
0.0      0.0      0.0      0.2      0.0      0.0      0.0      99.0
∗NMACH MACH
1.0      0.83
∗NALFA  ALPHA
1.0      3
∗LATRL  PSI      PITCHQ  ROLLQ   YAWQ    VINF
0.0      0.0      0.0      0.0      0.0      1.0
∗NPAN    SREF     CBAR     XBAR     ZBAR     WSPAN
09.0     3648.0   22.03    72.0     0.0      165.58
∗−−−−−−−−−−−−−−−−−−−−
∗X1−−−−−−− Y1−−−−−−−− Z1−−−−−−−−− CORD1−−−− COMMENT:
    FUSIFORM BODY
   0.00      0.00      9.875    192.42
   0.00      0.00     −9.875    192.42
∗NVOR    RNCV     SPC      PDL
   6.00     50.00     0.00     999.00
∗
∗ PHI   R0
  90.0   9.875
  60.0   9.875
  30.0   9.875
   0.0   9.875
 −30.0   9.875
 −60.0   9.875
 −90.0   9.875
∗AINC1      ANINC2  ITS  NAP IQUANT ISYNT NPP
   0.0000    0.0000    1.     10.     2.     0.     1.
∗
∗ SHAPING BODY
∗
∗ X/C (\%)
0.
2.5
5.
```

10.
30.
40.
50.
85
90.
100.
*
* CAMBER OF BODY AXIS
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
*
* AREA RATIO OF FUSIFORM BODY (\%)
0.
25.
50
100.
100.
100.
100.
50.
25.
10.
*
*————————————————————
* UPPER VERTICAL TAIL
* PANEL

*X1	Y1	Z1	CORD1	Comment:		
169.88	0.0	24.95	22.77			
189.40	0.0	45.55	10.84			
*NVOR	RNCV	SPC	PDL			
5.0	10.0	1.0	0.0			
*AINC1	ANINC2	ITS	NAP	IQUANT	ISYNT	NPP
0.0	0.0	0.0	0.0	1.0	0.0	0.0

```
*
*--------------------
* LOWER VERTICAL TAIL
* PANEL
*X1       Y1        Z1        CORD1    Comment:
161.93    0.0       9.875     23.49
161.93    0.0       15.56     23.49
*NVOR     RNCV      SPC       PDL
5.0       10.0      1.0       0.0
*AINC1    ANINC2    ITS       NAP      IQUANT    ISYNT     NPP
0.0       0.0       0.0       0.0      1.0       0.0       0.0
*
*--------------------
* HORIZONTAL TAIL
* PANEL
*X1       Y1        Z1        CORD1    Comment:
170.24    9.88      3.00      17.35
186.87    49.29     8.00      7.95
*NVOR     RNCV      SPC       PDL
10.0      5.0       1.0       0.0
*AINC1    ANINC2    ITS       NAP      IQUANT    ISYNT     NPP
0.0       0.0       0.0       0.0      0.0       0.0       0.0
*
*--------------------
* INBOARD MAIN WING + YEHUDI
* PANEL
*X1       Y1        Z1        CORD1    Comment:
79.5      9.88      0.367     39.4
96.51     31.57     1.172     25.29
*NVOR     RNCV      SPC       PDL
10.0      5.0       1.0       0.0
*AINC1    ANINC2    ITS       NAP      IQUANT    ISYNT     NPP
0.0       0.0       0.0       0.0      0.0       0.0       0.0
*
*--------------------
* OUTBOARD MAIN WING
* PANEL
*X1       Y1        Z1        CORD1    Comment:
96.51     31.57     1.172     25.29
136.5     82.79     3.07      9.4
*NVOR     RNCV      SPC       PDL
30.0      5.0       1.0       0.0
```

```
*AINC1   ANINC2  ITS       NAP       IQUANT ISYNT    NPP
0.0      0.0     0.0       0.0       0.0    0.0      0.0
*
*--------------------
*X1------- Y1-------- Z1--------- CORD1---- COMMENT:
    VERTICAL TAIL ENGINE
 157.59      0.00    24.95     036.87
 157.59      0.00    15.56     036.87
*NVOR    RNCV     SPC      PDL
   6.00    50.00     0.00    999.00
*
* PHI   R0
  90.0   4.695
  60.0   4.695
  30.0   4.695
   0.0   4.695
 -30.0   4.695
 -60.0   4.695
 -90.0   4.695
*AINC1      ANINC2  ITS  NAP IQUANT ISYNT NPP
   0.0000   0.0000   1.   10.    2.    0.   1.
*
* SHAPING BODY
*
* X/C (\%)
0.
2.5
5.
10.
30.
40.
50.
85
90.
100.
*
* CAMBER OF BODY AXIS
0.
0.
0.
0.
0.
```

```
0.
0.
0.
0.
0.
*
* AREA RATIO OF FUSIFORM BODY (\%)
100.
100.
100.
100.
100.
100.
100.
100.
100.
100.
*
*--------------------
*X1------- Y1-------- Z1--------- CORD1---- COMMENT:
    OUTER ENGINE
 075.83       26.83    0.295     011.90
 075.83       26.83    -9.28     011.90
*NVOR    RNCV      SPC       PDL
   6.00      50.00       0.00      999.00
*
* PHI   R0
  90.0   4.792
  60.0   4.792
  30.0   4.792
   0.0   4.792
 -30.0   4.792
 -60.0   4.792
 -90.0   4.792
*AINC1       ANINC2  ITS  NAP IQUANT ISYNT NPP
   0.0000     0.0000     1.      10.      2.      0.     1.
*
* SHAPING BODY
*
* X/C (\%)
0.
2.5
```

5.
10.
30.
40.
50.
85
90.
100.
*
* CAMBER OF BODY AXIS
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
*
* AREA RATIO OF FUSIFORM BODY (\%)
100.
100.
100.
100.
100.
100.
100.
100.
100.
100.
*
*————————————————————
*X1——————— Y1———————— Z1————————— CORD1———— COMMENT:
    INNER ENGINE
 075.83   26.83   −9.28   011.90
 075.83   26.83   0.295   011.90
*NVOR   RNCV   SPC   PDL
   6.00   50.00   0.00   999.00
*
* PHI   R0

```
  90.0   −4.792
  60.0   −4.792
  30.0   −4.792
   0.0   −4.792
 −30.0   −4.792
 −60.0   −4.792
 −90.0   −4.792
∗AINC1       ANINC2  ITS  NAP IQUANT ISYNT NPP
   0.0000     0.0000     1.     10.      2.      0.     1.
∗
∗ SHAPING BODY
∗
∗ X/C (\%)
0.
2.5
5.
10.
30.
40.
50.
85
90.
100.
∗
∗ CAMBER OF BODY AXIS
0.
0.
0.
0.
0.
0.
0.
0.
0.
0.
∗
∗ AREA RATIO OF FUSIFORM BODY (\%)
100.
100.
100.
100.
100.
```

100.
100.
100.
100.
100.
*
*——————————————————
* NXS NYS NZS
0.0 0.0 0.0
*
*
* END