Stanford University Unstructured (SU^2) User's manual

Francisco Palacios (Engineering Research Associate),
Juan J. Alonso (Associate Professor),
Karthik Duraisamy (Assistant Professor, Consulting),
Michael Colonno (Engineering Research Associate),
Jason Hicken (Postdoctoral Fellow),
Alejandro Campos, Sean Copeland, Thomas Economon,
Amrita Lonkar, Thomas Taylor
(Ph.D. Candidates)

susquared-dev@list.stanford.edu

Aerospace Design Laboratory (ADL)
Department of Aeronautics and Astronautics
Stanford University

January 19th, 2012

Abstract

The Stanford University Unstructured (SU^2) suite is an open-source collection of C++ based software tools for performing Partial Differential Equation (PDE) analysis and solving PDE constrained optimization problems. The toolset is designed with computational fluid dynamics and aerodynamic shape optimization in mind, but is extensible to treat arbitrary sets of governing equations such as potential flow, electrodynamics, chemically reacting flows, and many others. SU^2 is under active development in the Aerospace Design Lab (ADL) of the Department of Aeronautics and Astronautics at Stanford University, and is released under an open-source license.

The following documentation comprises an introduction to the suite, user and developer guides and the latest releases for downloading.

Contents

Chapter 1

Introduction

1.1 About

The SU^2 software suite specializes in high fidelity PDE analysis and design of PDE constrained systems on unstructured domains. The suite itself is composed of several C++ analysis modules that handle specific tasks, including the solution of the PDE system, decomposition of the domain for parallel computations, grid deformation, and many other tasks required for shape optimization and sensitivity studies. Some specifics of these modules can be found on the SU^2 Tools page of this documentation with more complete information available in the Developer's Guide. The current development of this code is done by the Aerospace Design Lab at Stanford University, based on the original structure of CADES 1.0 (2009 - 2010).

The software structure has been designed for maximum flexibility, leveraging the class-inheritance features native to the C++ programming language. This makes SU^2 an ideal vehicle for performing multi-physics simulations, including multi-species thermochemical non-equilibrium flow analysis, combustion modeling, two-phase flow simulations, magnetohydrodynamics simulations, etc. Additionally, the decomposition of the flow solver allows for the rapid implementation of new spatial discretization methods and time-integration schemes.

1.1.1 PDE Analysis

SU^2 is ideally suited to perform high fidelity, PDE analysis over complex geometries using unstructured mesh technology. The solver, SU2_CFD, is vertex-centered and finite-volume based. It currently offers the following schemes for spatially discretizing a range of governing equations:

- Jameson-Schmidt-Turkel or JST (centered scheme, second-order accurate in space).
- Lax-Friedrich (centered scheme, first-order accurate in space).
- Roe 1st-Order (upwind scheme, first-order accurate in space).
- Roe 2nd-Order (upwind scheme, second-order accurate in space using MUSCL scheme and Venkatakrishnan's limiter).

For time integration (explicit, and implicit), SU^2 offers the following schemes:

- Backward and forward Euler (first-order accurate in time).
- Runge-Kutta Explicit (up to fourth-order accurate in time).
- Dual time stepping (second-order accurate in time).

Other integration schemes are currently under development, and the class structure of SU^2 makes adding new schemes simple. Convergence acceleration and robustness enhancement features are available such as agglomeration multigrid schemes and residual smoothing. SU^2 can also execute in parallel for large-scale simulations using a Message Passing Interface (MPI) implementation. More details and specifics are available in the SU^2 Tools and the Developer's Guide.

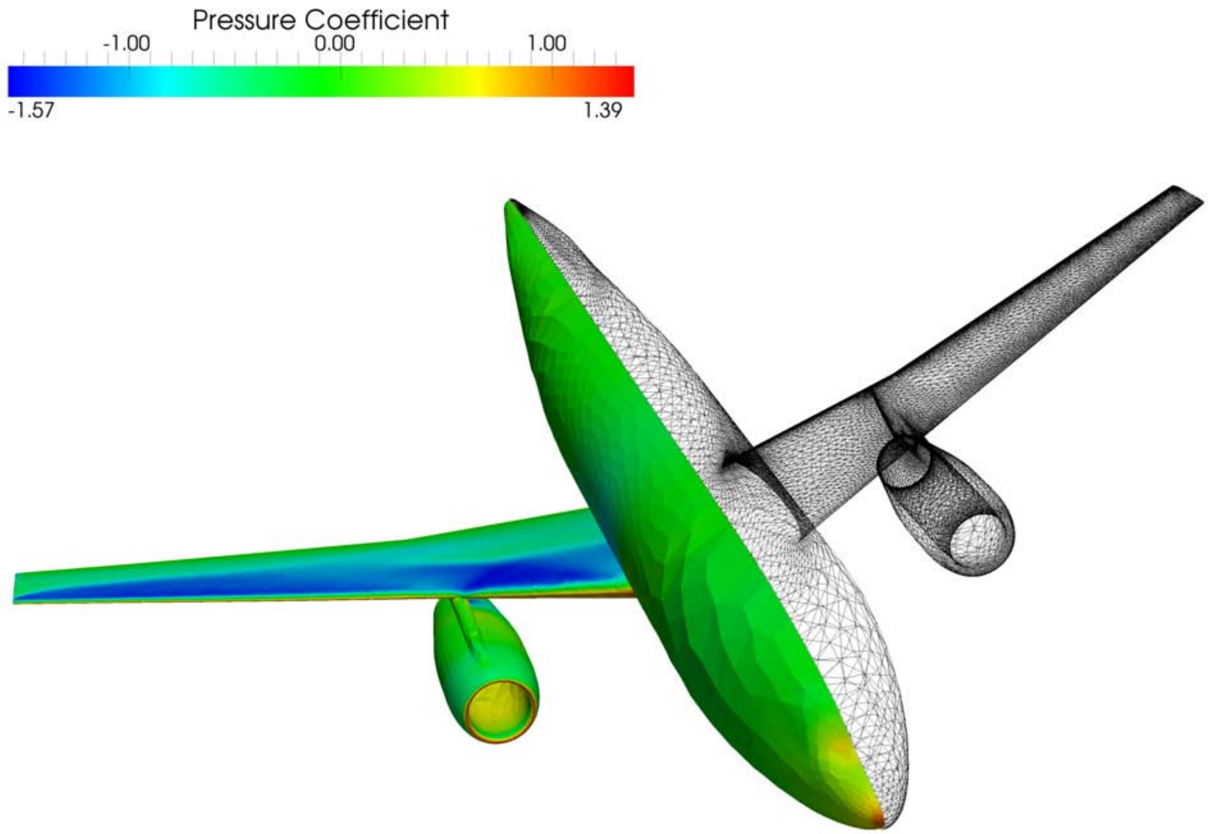


Figure 1.1: Pressure contours (left) and the unstructured surface mesh (right) on the DLR F6 wing-body configuration.

Reynolds Averaged Navier-Stokes (RANS), Laminar Navier Stokes & Euler Simulations

The current release of SU^2 comes pre-packaged with the Reynolds Averaged Navier-Stokes (with the Spalart-Allmaras turbulence model), laminar Navier-Stokes, and Euler equations (among others) to perform a wide range of general fluid dynamics analysis problems. Solutions from the software suite for a variety of configurations spanning the subsonic, transonic, supersonic, and hypersonic flight regimes are shown throughout this page to highlight the capabilities of the code.

Rotating Frame Simulations

A rotating frame formulation of the Euler equations is included in SU^2 for efficient, steady analysis of inviscid fluid around rotating aerodynamic bodies. Potential applications include wind turbines, turbomachinery, propellers, open-rotors, helicopter rotors, etc.

Multi-Physics Simulations

The general formulation of the solution vector in the CFD module permits the rapid implementation of additional governing equation sets and source terms to accommodate multi-physics simulations with ease. Results from a coupled electric potential, inviscid flow analysis is shown here for a multi-species Argon gas mixture with charge separation.

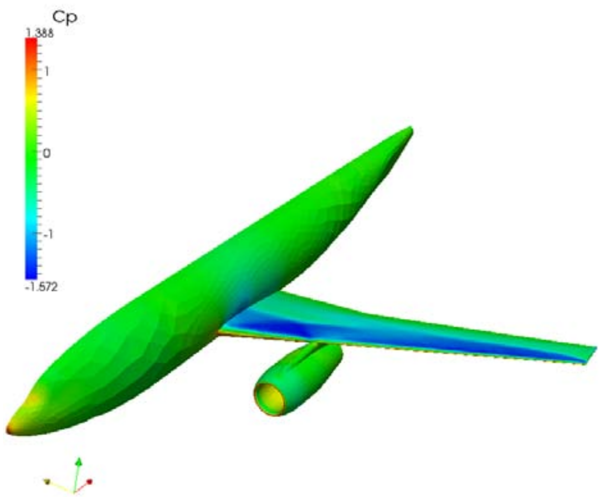


Figure 1.2: Euler solution of the DLR-F6 wing-body configuration.

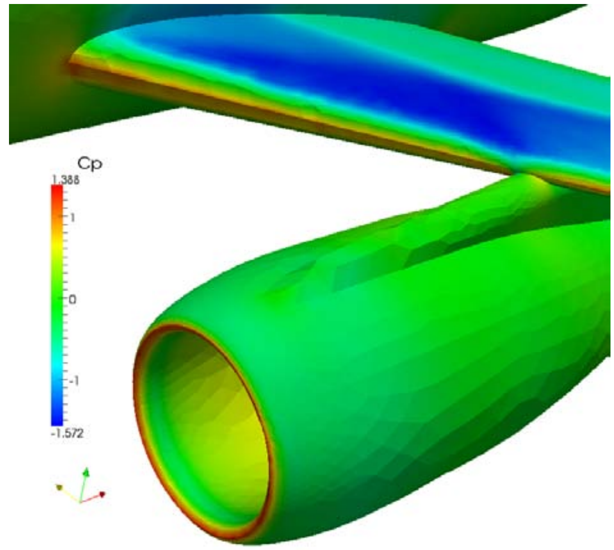


Figure 1.3: Transonic flow features around the nacelle of the DLR-F6.

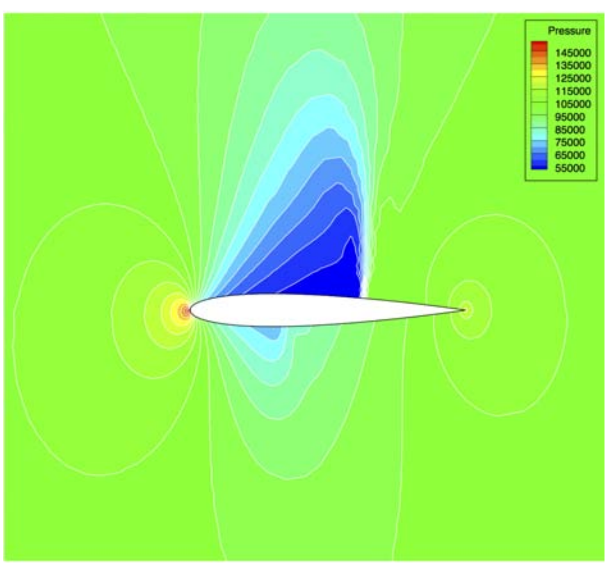


Figure 1.4: NACA 0012 pressure contours (Euler).

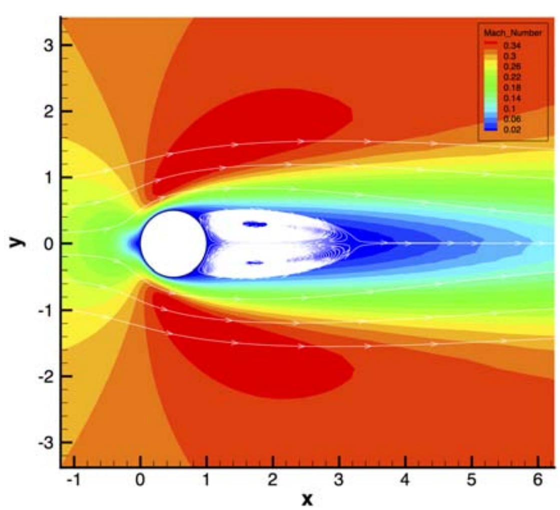


Figure 1.5: Mach contours and streamlines for viscous flow around a cylinder (Re = 40).

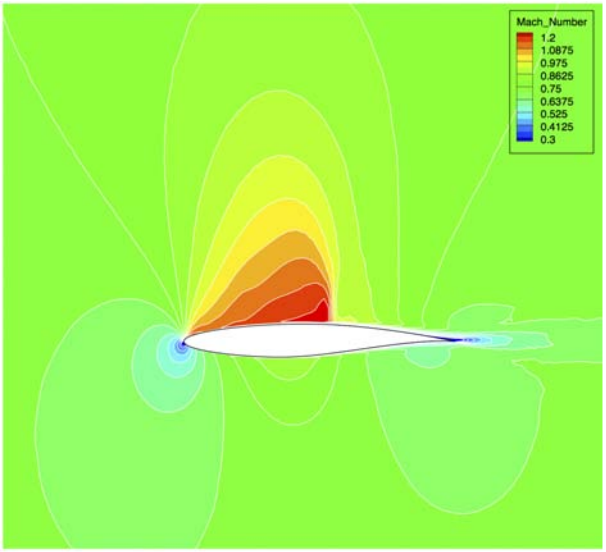


Figure 1.6: RAE 2822 mach number contours (RANS-SA).

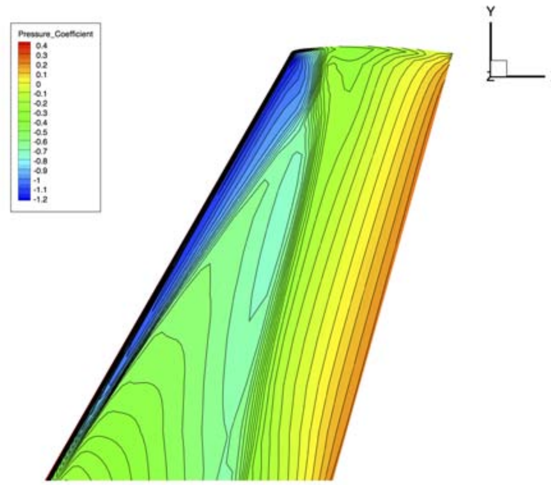


Figure 1.7: Pressure contours on the upper surface of the ONERA M6 wing (RANS-SA).

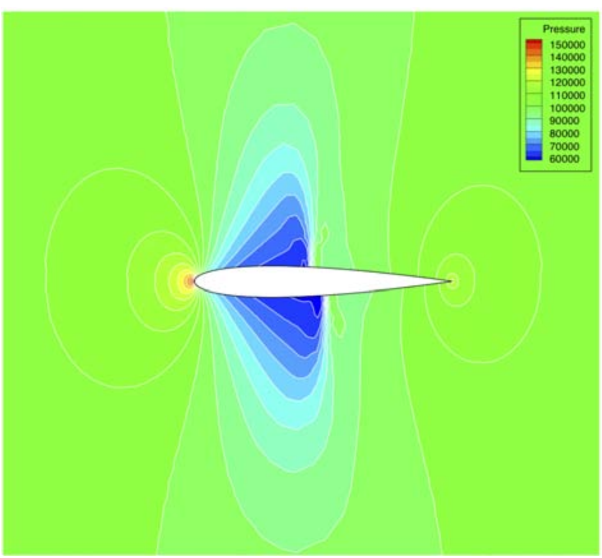


Figure 1.8: Pressure contours of a rotating NACA 0012 airfoil.

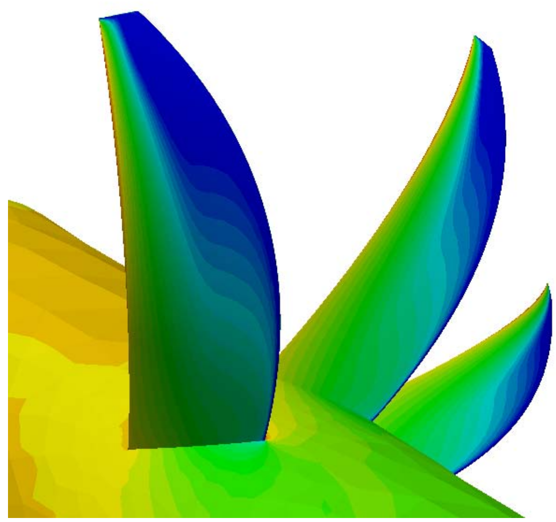


Figure 1.9: Pressure contours on the surface of a generic open rotor engine configuration.

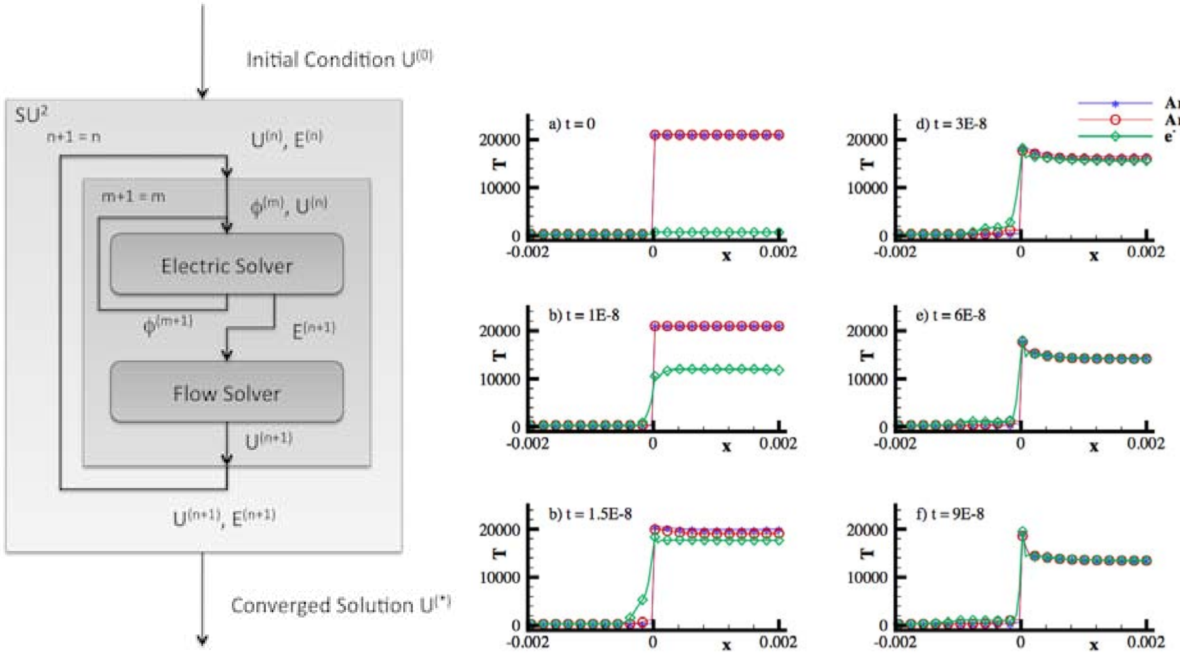


Figure 1.10: Time evolution of temperature in multi-species Argon plasma, passing through a Mach 15 shock wave, until thermal equilibrium is achieved.

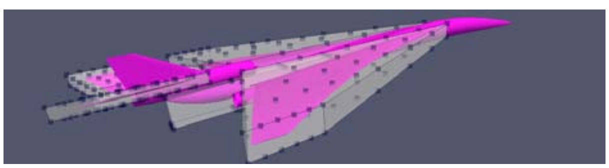


Figure 1.11: Top view of the Lockheed N+2 free-form deformation structure.

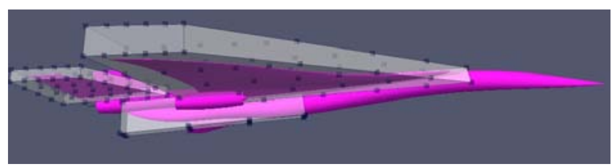


Figure 1.12: Lower view of the free-form deformation structure.

Optimal Shape Design

The design optimization of PDE constrained systems is a primary function of the SU^2 suite. The built-in adjoint solver in the CFD module, in conjunction with the Gradient Projection Code (GPC), and Mesh Deformation Code (MDC) deliver surface sensitivities and objective function gradients to Scipy-based optimization algorithms, enabling surface shape optimization for complex geometries. Furthermore, both the flow and adjoint solvers within SU^2 can be executed in parallel within the design loop for increased computational efficiency. Each of the design examples on this page take advantage of the Euler and Adjoint-Euler flow solvers in parallel with the design process driven by SciPy optimizers. Constraints such as a minimum lift or maximum moment can also be included.

Adaptive Mesh Refinement

The Mesh Adaptation Code (MAC) in the SU^2 suite facilitates strategic mesh adaptation based on several common schemes, including gradient and adjoint-based methods.

1.2 SU^2 Tools

The SU^2 software suite is composed of seven C++ based software modules that perform a wide range of PDE analysis activities. An overall description of each of module is included below to give perspective of the suite's capabilities, while more details can be found in the Developer's Guide. Some modules can be executed

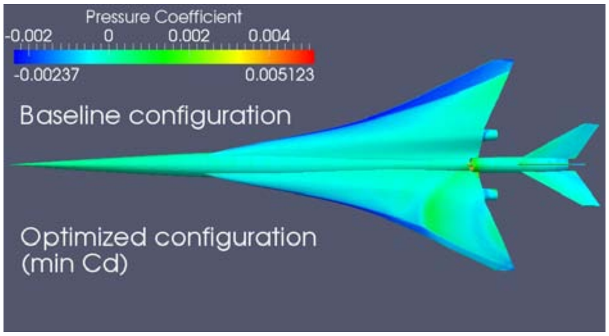


Figure 1.13: Baseline vs. optimized Cp contours for the vehicle upper surface.

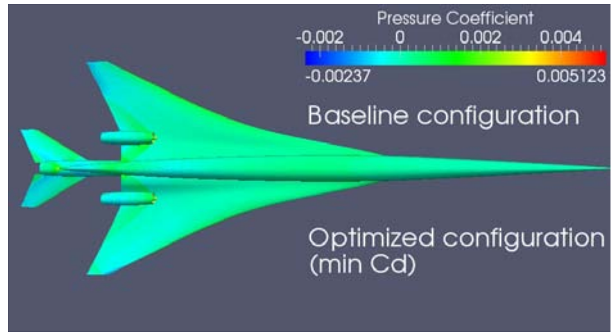


Figure 1.14: Baseline vs. optimized Cp contours for the vehicle lower surface.

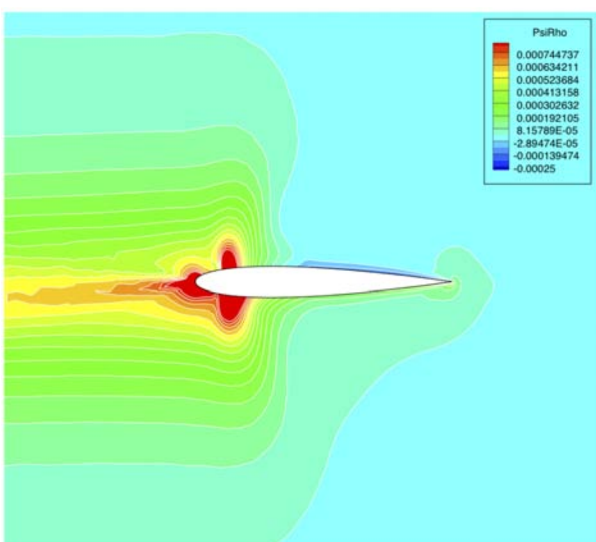


Figure 1.15: Adjoint density solution for the rotating NACA 0012 airfoil.

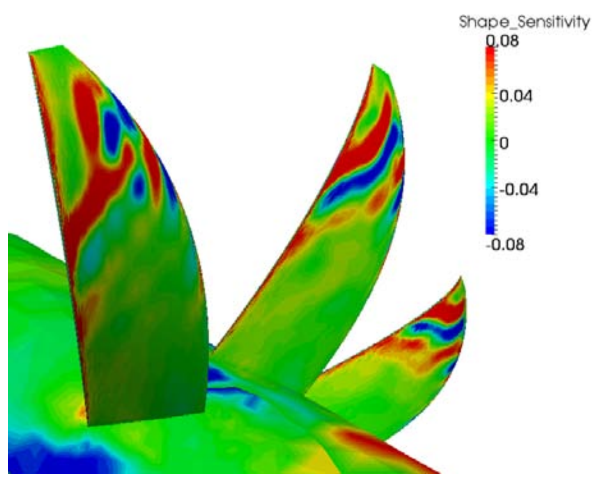
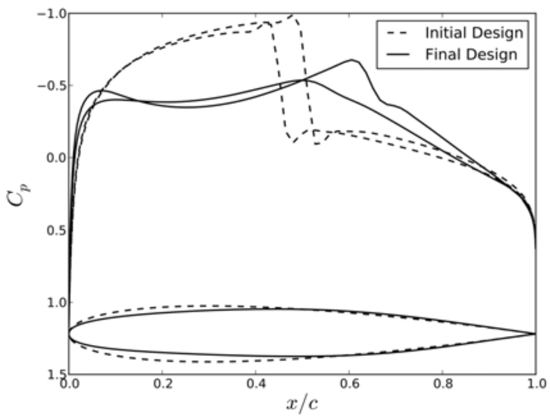


Figure 1.16: Surface sensitivity map for the open rotor configuration.



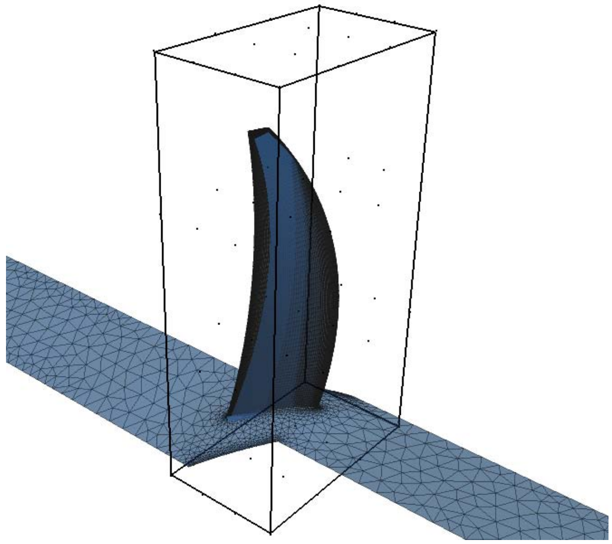


Figure 1.17: Initial and final profiles for the rotating airfoil shape design.

Figure 1.18: Free-form twist deformations for shape design of an open rotor blade.

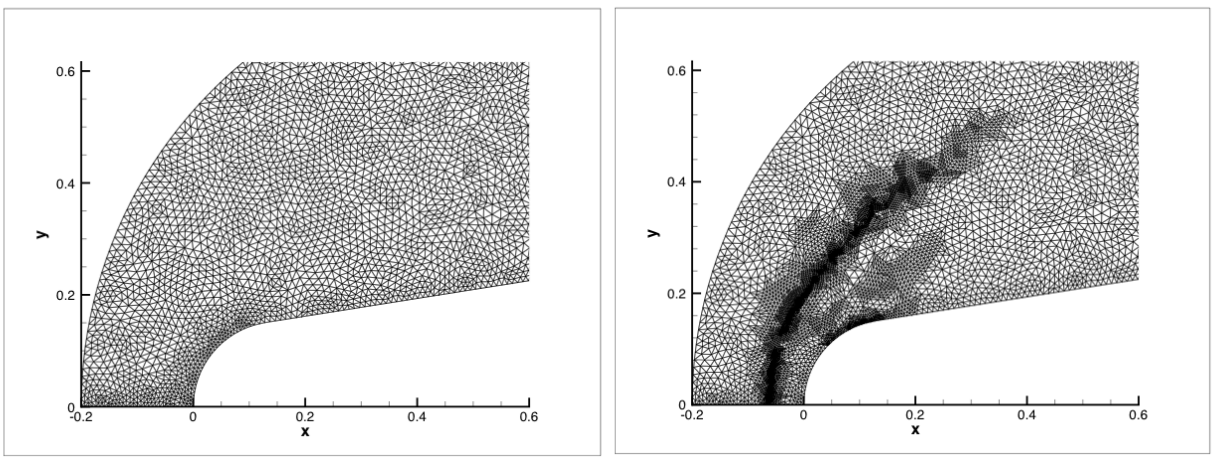


Figure 1.19: Adjoint-based mesh refinement for the RAM-C II hypersonic flight test experiment.

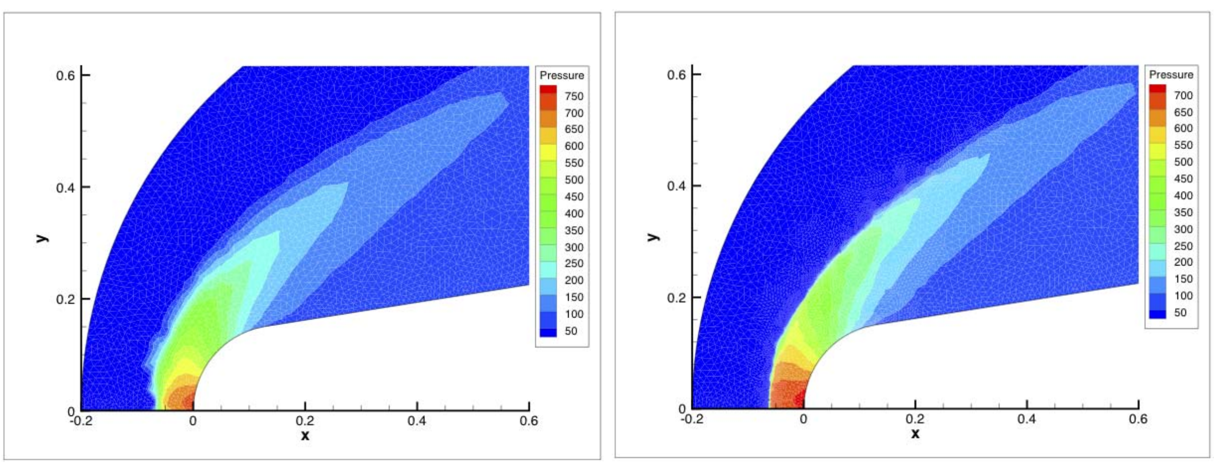


Figure 1.20: Adjoint-based mesh refinement for the RAM-C II hypersonic flight test experiment.

individually, most notably SU2_CFD, to perform high fidelity analysis, but the real power of the suite lies in the coupling of the modules to perform complex activities including design optimization and adaptive grid refinement among others. The SU^2 Tools section in the documentation showcases the major features of the current release of the software.

A key feature of the C++ modules is that each has been designed to separate functionality as much as possible and to leverage the advantages of the class-inheritance structure of the programming language. This makes SU^2 an ideal platform for prototyping new numerical methods, discretization schemes, governing equation sets, mesh perturbation algorithms, adaptive mesh refinement schemes, parallelization schemes, etc. You simply need to define a new subclass and get down to business. This philosophy makes SU^2 quickly extensible a wide variety of PDE analyses suited to the needs of the user, and work is ongoing to expand the features to incorporate additional features for future SU^2 releases. More information on feature additions can be found in the Future Developments section of the documentation.

1.2.1 C++ Software Modules

- SU2_CFD (Computational Fluid Dynamics Code) - Solves direct, adjoint and linearized problems for the Potential, Euler, Navier-Stokes and Reynolds-Averaged Navier-Stokes (RANS) equation sets. SU2_CFD can be run serially or in parallel using MPI. It uses a Finite Volume Method (FVM), and an edge based structure. Explicit and implicit time integration methods are available with centered or upwinding spatial integration schemes. The software also has several advanced features to improve robustness and convergence, including residual smoothing and agglomeration multigridding.

- SU2_GPC (Gradient Projection Code) - Computes the partial derivative of a functional with respect to variations in the aerodynamic surface. SU2_GPC uses the surface sensitivity, the flow solution, and the definition of the geometrical variable to evaluate the derivative of a particular functional (e.g. drag, lift, etc.).

- SU2_MDC (Mesh Deformation Code) - Computes the geometrical deformation of an aerodynamic surface and the surrounding volumetric grid. Once the type of deformation is defined, SU2_MDC performs the grid deformation using a spring-mass analogy. Three dimensional deformations use a method called Free Form Deformation, while two dimensional problems typically use bump functions, such as Hicks-Henne.

- SU2_MAC (Mesh Adaptation Code) - Performs grid adaptation using various techniques based on an analysis of a converged flow solution, adjoint solution and linearized problem to strategically refine the mesh about key flow features.

- SU2_DDC (Domain Decomposition Code) - Partitions the specified volumetric grid to be used by SU2_CFD when performing parallel computations. SU2_DDC requires grid partitioning software (i.e. METIS) to identify and assign nodes to each processor. Once that information is received, SU2_DDC prepares the communication between nodes and generates the appropriate computational grids (.dpl extensions) required for SU2_CFD to run in parallel.

- SU2_GDC (Geometric Design Code) - Computes the value of functional and the gradient(s) depending solely on the shape of the vehicle.

- SU2_PBC (Periodic Boundary Code) - Creates ghost cells in the computational domain for performing simulations with periodic boundary conditions. This module must be run prior to SU2_CFD for any simulation with periodic boundary conditions specified. Python Scripts

As described previously, the various software modules of SU^2 can be coupled together to perform complex analysis and design tasks using supplied Python scripts. A brief description of the scripts included in the current release of the software is provided below.

- continuous_adjoint.py - Automatically computes the sensitivities of a specified functional to design parameter perturbations (specified in the SU2_CFD configuration file) using continuous adjoint methods. The SU2_CFD and SU2_GPC modules are then called to perform the analysis. The script outputs a comma-separated (.csv) file containing the appropriate derivative information. finite_differences.py - Automatically computes the sensitivities of a specified functional to design parameter perturbations using a finite-difference method. As with the continuous_adjoint.py script, derivative information is read from the configuration file and SU2_CFD is called repeatedly to calculate the appropriate gradient elements. Outputs of finite_differences.py are also comma-separated files (.csv) containing the desired derivative information.

- grid_adaptation.py - Automates the grid adaptation procedure. The script links SU2_CFD and SU2_MAC, refining the input grid based on parameters specified in the configuration file. parallel_computation.py - Handles the setup and execution of parallel jobs on multicore or clustered computing architectures. The script calls SU2_DDC to partition the grid for the specified number of processors, then executes SU2_CFD in parallel.

- merge_solution_paraview.py - Merges the solutions obtained at each processor for parallel computations into a single solution file for the PareView software.

- merge_solution_tecplot.py - Merges the solutions obtained at each processor for parallel computations into a single solution file for the Tecplot software.

- merge_restart_SU2.py - Merges the internal restart files (.dat) obtained at each processor for parallel computations into a single restart file. divide_solution_SU2.py - Decomposes a single restart file (.dat) into multiple files for each processor of a parallel simulation.

- shape_optimization.py - Orchestrates all SU^2 modules to perform shape optimization. Objective function, design variables and additional module settings specifying the optimization problem are controlled through the configuration file.

1.3 Future Developments

Developments currently being pursued for the next release of SU^2 (tentatively scheduled for Summer 2012) include:

- Extended CGNS support (more mesh input capability, solution output and parallel support)
- SST turbulence model (RANS)
- Axisymmetric implementations (Euler and Reacting NS)
- Discrete adjoint methods for the Euler equations
- Rotating frame implementation for the RANS equations
- Mesh adaptation feature expansion (for parallel calculations and multiple cell types)
- GMRES/Newton-Krylov solver for the flow equations
- Multi-species viscous flows (Reacting NS)
- Thermochemical non-equilibrium models (Reacting NS)
- 2-D implementation (Reacting NS)
- Adding support for multiple optimizers for optimal shape design

If you have suggestions or would like to request new features please contact the developers. Any developers adding capability to SU^2 are also encouraged to contribute their new additions to the project.

Chapter 2

Download & License

2.1 Download

2.1.1 Downloading SU^2

The SU^2 software suite version 1.0 is available for download under the GNU General Public License (GPL) v3. Please reference the License page for terms and conditions.

- Source code tar files are available for download, along with pre-compiled binary executables for selected platforms.
- Test cases (configuration and mesh files) are included in the source code tar file.
- Release Notes are available for major and interim releases.
- The development team looks forward to your feedback. Please join the SU^2 community by subscribing to the user's mailing list to receive important updates about the code, and send your feedback directly to the developers through the developer's mailing list. See the Contact page for details on the mailing lists.

2.1.2 Current Stable Release: SU^2 Version 1.0

Available distributions:

- Binaries.
  - Mac OS X: Gnu C++ compiler (gcc 4.2.1, Apple Inc. build 5666, dot 3).
  - Linux (Red Hat): Gnu C++ compiler (gcc 4.1.2 20080704, Red Hat 4.1.2-51).
  - Linux (Red Hat): Intel C++ compiler (icc 12.0.2 20110112).
  - Windows XP: Windows C++ compiler (Microsoft Windows Resource Compiler Version 6.1.6723.1).
- Source Code.

Are you a Windows user installing from source? See notes section on the installation page.

Note I: Binary distributions have reduced functionality. All optional libraries and dependencies are removed. To run CGNS meshes and parallel computations, you must download the source code and compile with the appropriate optional dependencies described fully in the Installation page.

Note II: SU^2 has been validated using the following C++ compilers: gcc 4.2.1, gcc 4.1.2, and icc 12.0.2. If you identify any problem using another C++ compiler, please let us know as soon as possible.

2.1.3 Register & Download

To download SU^2, please take a few seconds and register by filling out the form on the page linked below. At the completion of the registration page, you will be re-directed to the appropriate distribution of the software. If you have already registered with us, please proceed to the URL provided to you when you completed the registration process. The information collected is used ONLY for usage statistics by the ADL development team. You will not be contacted for any reason, unless you choose to join the susquared-users mailing list. Your cooperation is greatly appreciated and lets us build a better tool to suit the needs of our community. Thanks!

2.1.4 Previous Releases

Version 1.0 is the first public release of the SU^2 suite. In the future, previous releases will be found here.

2.1.5 Development Releases

For active members of the SU^2 development team, the most current version of the software package is available on the Aerospace Design Laboratory's Subversion software repository. Contact the ADL server administrators if you would like to request access to the repository.

2.1.6 Third-party Downloads

The following is a list of some third-party software packages that extend the capability of SU^2. Each is freely available and is linked to the corresponding download site. Much more information on these packages can be found on the Installation page.

- ParaView (data visualization package)
- Python 2.6.6 (scripts for automating the SU^2 tools)
- CGNS 3.1.3 (allows the use of third-party meshing software that exports the CGNS format)
- METIS 4.0.3 (graph partitioning for running simulations in parallel, see installation page for more detail)
- OpenMPI or MPICH2 (MPI implementation for running simulations in parallel)

2.2 License

Skip to end of metadata Added by Sean Copeland, last edited by Thomas Taylor on Jan 06, 2012 (view change) show comment Go to start of metadata The Stanford University Unstructured (SU^2) Code is copyright protected under the GNU General Public License (GPL) v3. Additional information can be found here.

2.2.1 Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

2.2.2 Terms and Conditions

0. Definitions.

This License refers to version 3 of the GNU General Public License.

Copyright also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

The Program refers to any copyrightable work licensed under this License. Each licensee is addressed as you. Licensees and recipients may be individuals or organizations.

To modify a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a modified version of the earlier work or a work based on the earlier work.

A covered work means either the unmodified Program or a work based on the Program.

To propagate a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To convey a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays Appropriate Legal Notices to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The source code for a work means the preferred form of the work for making modifications to it. Object code means any non-source form of a work.

A Standard Interface means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The System Libraries of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A Major Component, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The Corresponding Source for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

1. The work must carry prominent notices stating that you modified it, and giving a relevant date.

2. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to keep intact all notices.
3. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.
4. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an aggregate if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

1. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.
2. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.
3. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.
4. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.
5. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A User Product is either (1) a consumer product, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, normally used refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

Installation Information for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

Additional permissions are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

1. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

2. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

3. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

4. Limiting the use for publicity purposes of names of licensors or authors of the material; or

5. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

6. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered further restrictions within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this

License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An entity transaction is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A contributor is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's contributor version.

A contributor's essential patent claims are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, control includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a patent license is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To grant such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. Knowingly relying means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is discriminatory if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License or any later version applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM AS IS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

2.3 Release Notes

2.3.1 Computational Fluid Dynamics Module (SU2_CFD)

Different options described in the tutorials. Serial, and parallel execution.

2.3.2 Domain Decomposition Module (SU2_DDC)

Computational domain decomposition is performed using METIS and is currently serial execution only.

2.3.3 Geometric Design Module (SU2_GDC)

Serial execution only.

2.3.4 Gradient Projection Module (SU2_GPC)

Serial execution only.

2.3.5 Mesh Adaptation Module (SU2_MAC)

Mesh adaptation is available for unstructured grids composed of triangular (2D) and tetrahedral (3D) cells for serial computations only. Only SU^2-native grid files can be used with SU2_MAC, so any CGNS files must first be converted using the software's built-in features. The following adaptation strategies are available:

- Full - Refines the full grid using homothetic subdivisions of the cells.
- Full Flow - Performs a full, homothetic subdivision of the grid and interpolates the converged flow solution onto the new mesh.
- Full Adjoint - Performs a full, homothetic subdivision of the grid and interpolates the converged adjoint solution onto the new mesh.
- Gradient Flow - Adds cell density to the baseline grid where large flow gradients are observed.
- Gradient Adjoint - Adds cell density to the baseline grid where large adjoint gradients are observed.
- Computable - Performs adjoint-based adaptation in accordance with Park's (2002) method for determining the computable correction the the grid error. Mesh Deformation Module (SU2_MDC)
- Free-form deformation for 3D design. Corresponding grid deformation is determined using either a spring analogy solved via a Conjugate Gradient (CG) method or an algebraic method. Serial execution only.

2.3.6 Periodic Boundary Module (SU2_PBC)

Serial execution only.

2.4 Known Issues

This page contains known issues with SU^2 listed by version number of the code. Each of these issues will be fixed in subsequent releases, and a solution for the problem, if available for the current release, is also given here. Please report any issues or bugs to the developers.

2.4.1 Version 1.0

- (2012.01.20) Data within a CGNS file (in particular the node coordinates) must be exported from a meshing package in double precision. CGNS meshes in single precision will be read by SU^2, but they will immediately result in NaNs for the residual updates upon starting a simulation.
- (2012.01.26) When compiling with CGNS support, SU^2 will throw an error if the CGNS library was built with HDF5. For Version 1.0 of SU^2, please build the CGNS library without HDF5.

Chapter 3

Quick Start Tutorial

3.1 Introduction

Welcome to the Quick Start Tutorial for the SU^2 software suite. This tutorial is intended to demonstrate some of the key features of the analysis and design tools in an easily accessible format. Completion of this tutorial only requires a few minutes. If you haven't done so already, please visit the Download and Installation pages to obtain the most recent stable release of the software and details for installation. This tutorial requires only the SU2_CFD tool from the SU^2 suite.

3.1.1 Goals

Upon completing this simple tutorial, the user will be familiar with performing the flow and adjoint simulation of external, inviscid flow around a 2-D geometry and able to plot the sensitivities along that surface. The specific geometry chosen for the tutorial is the NACA 0012 airfoil. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 2-D, Euler and Continuous Adjoint Euler equations
- Multigrid
- JST numerical scheme in space
- Euler implicit time integration
- Euler Wall and Farfield boundary conditions

3.1.2 Resources

After obtaining a copy of the SU2_CFD flow solver, two other files are needed as input to the code: a configuration file describing the options for the particular problem, and the corresponding computational mesh file. These configuration and mesh files are also available in the SU2/TestCases/inv_NACA0012/ directory provided with the code:

- NACA0012 config file
- NACA0012 mesh (SU^2 native format)

These sample results files from SU^2 can be used to compare the results you obtain from the tutorial:

3.1.3 Tutorial

The following tutorial will walk you through the steps required when computing the flow and adjoint solutions around the NACA 0012 airfoil using SU^2. Again, it is assumed that you have already obtained and compiled the SU2_CFD code (either individually, or as part of the complete SU^2 package) for a serial computation. If you have yet to complete this requirement, please see the Download and Installation pages.

Tecplot Format:	ParaView Format:
flow.plt	flow.vtk
surface_flow.plt	surface_flow.vtk
surface_flow.csv	surface_flow.csv
history_flow.plt	history_flow.csv
adjoint.plt	adjoint.vtk
surface_adjoint.plt	surface_adjoint.vtk
surface_adjoint.csv	surface_adjoint.csv
history_adjoint.plt	history_adjoint.csv

3.1.4 Background

The NACA0012 airfoil is one of the four-digit wing sections developed by the National Advisory Committee for Aeronautics (NACA), and is a widely used geometry for wings. The numbering system is such that the first number indicates the maximum camber (percent of chord), the second shows the location of the maximum camber (tens of percent of chord) and the last two digits indicate the maximum thickness (percent of chord). More information on these airfoil sections can be found here or in the book 'Theory of Wing Sections' by Abbott and von Doenhoff.

3.1.5 Problem Setup

This problem will solve the Euler equations on the NACA0012 airfoil at an angle of attack of 1.25°, using air with the following freestream conditions:

- Pressure = 101325 N/m^2
- Temperature = 273.15 K
- Mach number = 0.8.

The aim is to find the flow solution and the adjoint solution with respect to an objective function defined as the drag over the airfoil.

Mesh Description

The unstructured mesh provided is in the .su2 format. It consists of 10,216 triangular cells, 5,233 points, and two boundaries named airfoil and farfield. The airfoil surface uses a flow-tangency Euler wall boundary condition, while the farfield uses a standard characteristic-based boundary condition. Figure (1) shows the computational mesh. Find more information on the supported mesh formats here.

Configuration File Options

Aside from the mesh, the only other file required to run the SU2_CFD solver details the configuration options. It defines the problem, including all options for the numerics, flow conditions, multigrid, etc., and also specifies the names of the input mesh and output files. In keeping simplicity for this tutorial, only two configuration options will be discussed. For a full explanation of the config file, please read the detailed explanation page.

Upon opening the inv_NACA0012.cfg file in a text editor, one of the early options is the MATH_PROBLEM:

```
%
% Mathematical problem (DIRECT, ADJOINT, LINEARIZED,
% ONE_SHOT_ADJOINT)
MATH_PROBLEM= DIRECT
```

SU^2 is capable of running the direct and adjoint problems for several sets of equations. The direct analysis solves for the flow around the geometry, and quantities of interest such as the lift and drag coefficient on the body will be computed. Solving the adjoint problem leads to an efficient method for obtaining the change in a single objective function (e.g. drag coefficient) relative to a large number of design variables (surface deformations). The direct and adjoint solutions often couple to provide the objective analysis and gradient

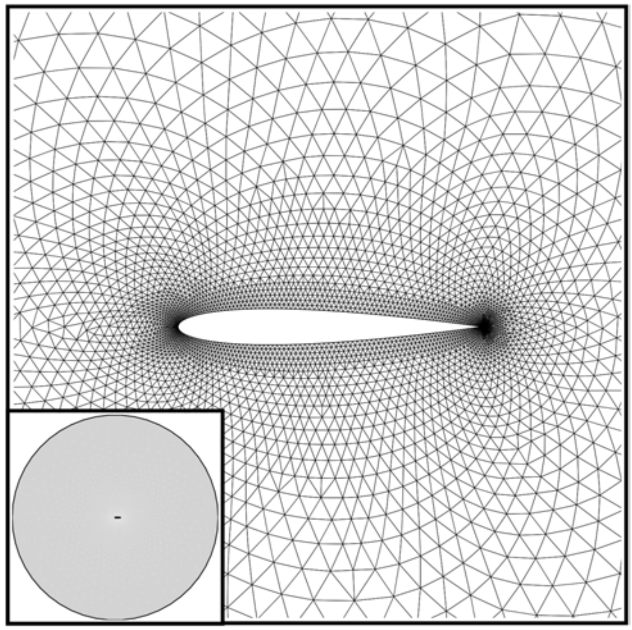


Figure 3.1: Far-field and zoom view of the computational mesh.

information needed by an optimizer when performing aerodynamic shape design. In this tutorial, we will perform DIRECT and ADJOINT solutions for the NACA 0012 airfoil.

The user can also set the format for the solution files:

```
%
% Output file format (PARAVIEW, TECPLOT)
OUTPUT_FORMAT= TECPLOT
```

SU^2 can output solution files in the .vtk and .plt formats which can be opened in the ParaView and Tecplot visualization software packages, respectively. We have set the file type to TECPLOT in this tutorial by default, but users without access to Tecplot are encouraged to download and use the freely available ParaView package. To output solution files for ParaView, set the OUTPUT_FORMAT option to PARAVIEW. Details on the visualization packages can be found on the installation page.

3.1.6 Running SU^2

Flow solution

The first step in this tutorial is to solve for the flow:

Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/directory. Copy the config file (inv_NACA0012.cfg) and the mesh file (mesh_NACA0012_inv.su2) to this directory. Run the executable by entering "./SU2_CFD inv_NACA0012.cfg" at the command line. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria. Files containing the flow results (with "flow" in the file name) will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt). More specifically, these files are:

- flow.plt or flow.vtk - full volume flow solution.
- surface_flow.plt or surface_flow.vtk - flow solution along the airfoil surface.
- surface_flow.csv - comma separated values (.csv) file containing values along the airfoil surface.
- restart_flow.dat - restart file in an internal format for restarting this simulation in SU2.
- history.plt or history.csv - file containing the convergence history information.

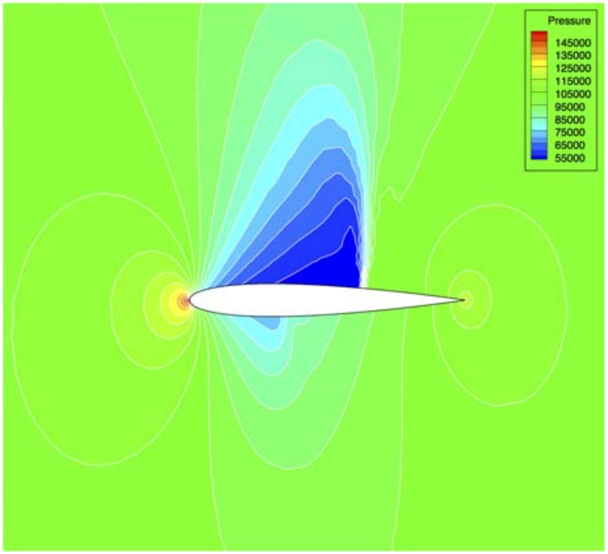


Figure 3.2: Pressure contours around the NACA 0012 airfoil.

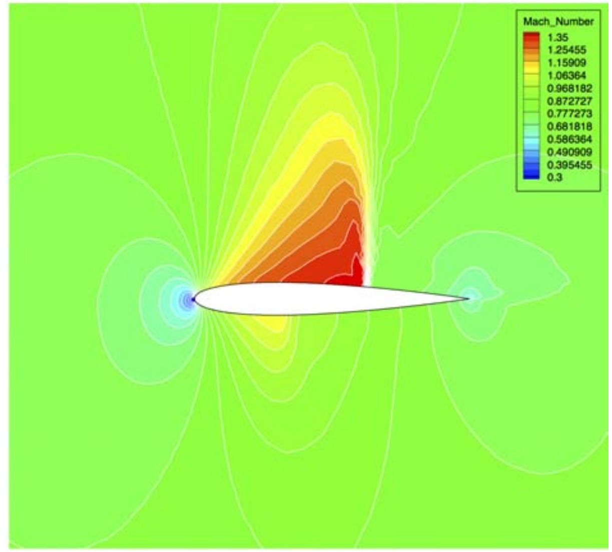


Figure 3.3: Mach number contours around the NACA 0012 airfoil.

Adjoint solution

Next we want to run the adjoint solution to get the sensitivity of the objective function (the drag over the airfoil) to conditions within the flow:

Open the config file and change the parameter MATH_PROBLEM from DIRECT to ADJOINT, and save this file. Rename the restart file (restart_flow.dat) to "solution_flow.dat" so that the adjoint code has access to the direct flow solution. Run the executable again by entering "./SU2_CFD inv_NACA0012.cfg" at the command line. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria. Files containing the adjoint results (with "adjoint" in the file name) will be written upon exiting SU2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt). More specifically, these files are:

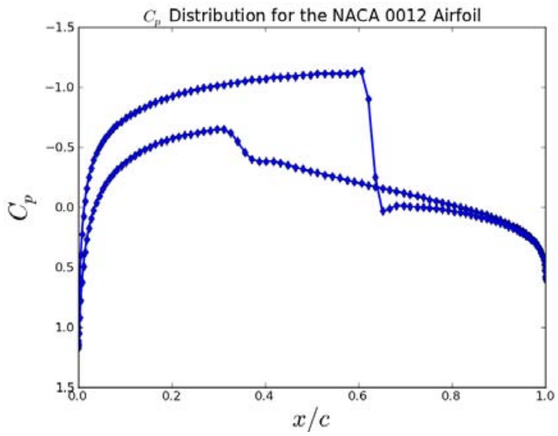
- adjoint.plt or adjoint.vtk - full volume adjoint solution.
- surface_adjoint.plt or surface_adjoint.vtk - adjoint solution along the airfoil surface.
- surface_adjoint.csv - comma separated values (.csv) file containing values along the airfoil surface.
- restart_adj_cd.dat - restart file in an internal format for restarting this simulation in SU2. Note that the name of the objective appears in the file name.
- history.plt or history.csv - file containing the convergence history information.

3.1.7 Results

The following figures were created in Tecplot using the SU^2 results. These results are contained in the flow.plt, surface_flow.csv, history.plt, adjoint.plt, and surface_adjoint.csv files.

Flow Solution

Adjoint Solution



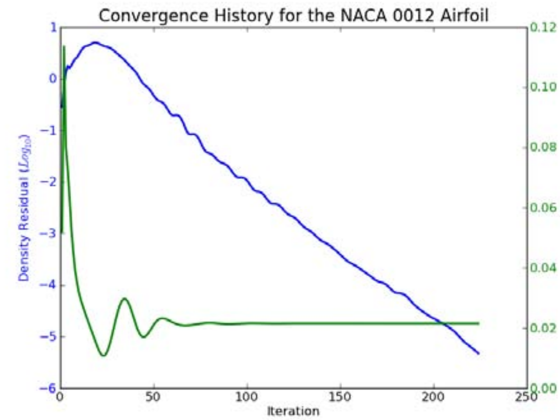
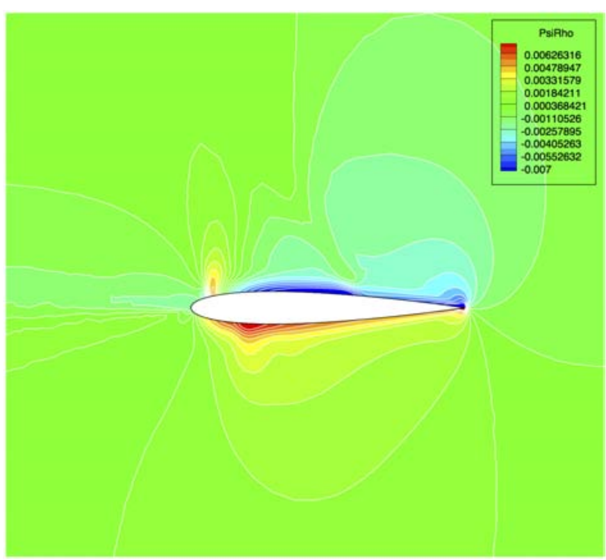


Figure 3.4: Coefficient of pressure distribution along the airfoil surface. Notice the strong shock on the upper surface (top line) and a weaker shock along the lower surface (bottom line).

Figure 3.5: Convergence history of the density residual and drag coefficient.



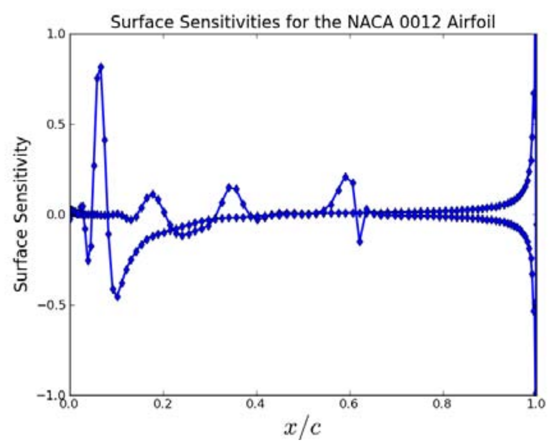


Figure 3.6: Contours of the adjoint density variable.

Figure 3.7: Surface sensitivities. The surface sensitivity is the change in the objective function due to an infinitesimal deformation of the surface in the local normal direction. These values are calculated at each node on the airfoil surface from the flow and adjoint solutions at negligible computational cost.

Chapter 4

User's Guide

Welcome to the User's Guide for the Stanford University Unstructured (SU2) software suite. This guide provides a general overview of the procedures required to Download and Install the software, run SU^2 from the command line, a description of input/output filetypes and contains several User's Tutorials that step through the process of utilizing the main features of the toolkit outlined in the SU^2 Tools. This guide does not give the details of the implementation and structure of the source code, however, that information is available in the SU^2 Developer's Guide. This guide is ideal for new users looking to do analysis and design, using features already implemented in the software.

Any deficiencies or requests for clarification in this guide can be reported to the SU^2 development team. Your feedback is greatly appreciated.

4.1 Installation

SU^2 has been designed with ease of installation and use in mind. This means that, wherever possible, a conscious effort was made to develop in-house solutions for code components rather than relying on third party packages or libraries. In many cases, the flow solver can be compiled and executed needing only a free C++ compiler and a way in which to visualize the results. However, the capabilities of SU^2 can be extended by obtaining some external software. For example, parallel computations will require an installed implementation of MPI (Message Passing Interface) for data communication across nodes. Again, to facilitate ease of use and to promote the open source nature, whenever external software is required within the SU^2 suite, packages that are free or open source have been favored. These dependencies and third party packages are discussed below.

4.1.1 Required Software

Command Line Terminal

In general, all SU^2 execution will occur via command line arguments within a terminal. For Unix/Linux or Mac OS X users, the native terminal applications will be needed. For Windows users, Cygwin is recommended.

Compilers

Installing SU^2 from source requires a C++ compiler. Gnu gcc/g++ compilers are open-source, widely used, and reliable for building SU^2. The Intel compiler set has been optimized to run on Intel hardware and has also been used successfully by the development team to build the source code, though it is commercially licensed. Academic licenses for the Intel compiler set are affordable for those who qualify.

- Gnu gcc / g++
- Intel icpc

Grid Generation

Users wishing to perform analyses on their own meshes must have a means of generating an appropriate computational domain. The native SU^2 grid format is designed for readability and ease of use, so users with simple mesh configurations can write scripts to generate the appropriate meshes. For more complex configurations, grid generation software is recommended (with capability to export in CGNS file format). Several open source and commercial products are available, and a list of those used by the development team are included below.

- Pointwise/Gridgen
- Gambit

Data Visualization

Users of SU^2 require a data visualization tool to post-process solution files. The software currently supports .vtk and .plt output formats natively read by ParaView and Tecplot, respectively. ParaView provides full functionality for data visualization and is freely available under an open source license. Tecplot is a commercially licensed software package widely used by the scientific computing community and is available for purchase.

- ParaView
- Tecplot

4.1.2 Binary Executable Installation

The binary executables available on the Download page are intended to get new users up and running as quickly as possible. This option is best for novice CFD users or those looking to quickly sample some of the features of the software tool. To make these binary distributions available to a wide range of potential users, some advanced features requiring external libraries and dependencies have been disabled (most notably the ability to run simulations in parallel). More specifically, the pre-compiled binary executables are simply the serial version of the SU2_CFD flow solver.

No installation is required for users who select the binary distribution of SU^2 (aside from the prerequisites listed above). Simply open a terminal and proceed to the Quick Start Tutorial.

4.1.3 Compiling from Source

Users looking to install from the source code are strongly encouraged to use the python build script included in the distribution. Component-by-component compilation is possible using standard makefiles on the command line with your C++ compiler of choice.

Automated Build Script

The python-based build script for the SU^2 software suite facilitates rapid installation of the package and operates in much the same way as a standard Makefile. The build script allows users the flexibility of specifying desired build features, including operating system, compiler, parallelization, CGNS libraries, etc., while hiding the details of the build process from the user. In principle, this maintains a user-friendly environment for first time installers and SU^2 developers. Step-by-step instructions for using the build script, and details about the various options available for specification at compile-time are below.

- In a terminal window, navigate to the SU2_HOME/SU2Py directory.
- A listing of the various build options is available by typing python ./build_SU2.py –help on the command line.
- Identify the appropriate flags from the build_SU2.py help documentation for your system architecture and desired functionality.
- Build the software by issuing a python ./build_SU2.py -YOUR -OPTION -FLAGS command.

- Note that the only required option for the script is the type of operating system.

Several examples:

- Full SU2 suite for Redhat Linux: python ./build_su2.py -o redhat
- Full SU^2 suite for Mac OS X operating systems using the Intel C++ compiler (if available) without parallelization: python ./build_SU2.py -o macosx –compiler=icpc
- SU2_CFD Module alone for Mac OS X with parallelization using an MPI-wrapped compiler (if available): python ./build_SU2.py -o macosx -p –SU2_CFD
- Full SU^2 suite for Redhat Linux operating systems, using gcc (default), with parallelization, and with the CGNS Libraries (note the path directly to the library file must be specified): python ./build_SU2.py -o redhat -p –with-cgns –cgns-inc-path=-I/usr/local/include –cgns-lib-path=/usr/local/lib/libcgns.a
- Cleaning the entire software suite on Mac OS X: python ./build_su2.py -o macosx -c
- Cleaning the SU2_CFD module alone on Redhat Linux: python ./build_su2.py -o redhat -c –SU2_CFD

A full listing of the build_SU2.py option flags are as follows:
Usage:

- $ python build_SU2.py [options]

Options:

- -h, –help show this help message and exit
- -o OS, –os=OS specify build OS (macosx, redhat)
- –compiler=CXX specify C++ compiler other than default (optional)
- -p, –parallel build parallel version
- -c, –clean clean all directories

CGNS Options:
Build with CGNS using these options.

- –with-cgns build with CGNS support
- –cgns-inc-path=-I/path/to/cgns/include specify path to CGNS header
- –cgns-lib-path=/path/to/libcgns.a specify path to CGNS library

SU^2 Suite Components:
Build or clean individual components rather than the entire suite by specifying one or more of the following build flags.

- –SU2_CFD build SU2_CFD
- –SU2_DDC build SU2_DDC
- –SU2_GDC build SU2_GDC
- –SU2_GPC build SU2_GPC
- –SU2_MAC build SU2_MAC
- –SU2_MDC build SU2_MDC
- –SU2_PBC build SU2_PBC

Assuming no compile-time errors, you are now ready to run your first simulation. Please proceed to the Quick Start Tutorial.

Manual Compilation Using Makefiles

Each of the SU^2 suite components can be compiled using standard makefiles within the terminal. Users seeking to install the software using this approach are assumed to have some proficiency with command line terminals and building software from source. The procedure for building each tool is the same, and can be achieved by following the steps outlined below:

- Set the $SU2_HOME environment variable to point toward the root directory of the software (i.e. the directory containing the license agreement and the various sub-directories for the software modules).
- Navigate to the config directory of the desired tool (e.g. $SU2_HOME/SU2_CFD/config for the SU2_CFD module) and identify the appropriate makefile for your operating system. If you do not see your operating system, you will have to generate your own. If create a new makefile for a specific OS, please visit the Contact page and share your makefile with the development team so that we can make it available to the community.
- Copy the appropriate makefile to the module root directory for the selected tool (e.g. $SU2_HOME/SU2_CFD for the CFD analysis module) and rename the OS-specific makefile to "makefile.in".
- Move back into the root directory for that tool. You should now have both a makefile and a makefile.in in that directory.
- Issue a "make all" command to build the software.
- Upon successful compilation, SU^2 will place a copy of the binary in the bin directory for that tool (e.g. $SU2_HOME/SU2_CFD/bin for the CFD analysis module) and also in the $SU2_HOME/SU2Py/ directory.
- To delete the compiled objects for a tool, type "make clean" in its root directory.

4.1.4 Notes for Windows Users

While we work on binaries for distribution, it is recommended that Windows users work with Cygwin - a terminal emulator with built-in GNU compilers and Python.

1. Download Cygwin here.
2. Run setup.exe and follow the install instructions (pages for install from internet, root directory, mirror, etc.)
3. Once on the Select Packages screen, click on the word "default" so that it changes to "install" for the following packages: Download the source code tar file, just as you would for Mac OS X/Redhat
   (a) python (to enable Python)
   (b) devel (to enable development tools like make and compilers)
4. Open Cygwin (by default, executing C:
   cygwin
   Cygwin.bat will open a new terminal)
5. Navigate to the location of the downloaded SU2source filesUnzip the source code files by entering "tar -xzf SU2vX.X.tgz" which will result in a new directory called SU2vX.X/, where the X.X is the current version number
   (a) to get to your C: drive, enter "cd /cygdrive/c" at the command line
   (b) for example, if you downloaded the source files into C:
   SU2, you could get to the directory by entering "cd /cygdrive/c/SU2"
6. Move into the SU2vX.X/SU2Py/ directory
7. Run the build script by entering "python ./build_SU2.py -o redhat –SU2_CFD". Note that your compiled binaries may have the extension ".exe" (e.g. SU2_CFD.exe) SU^2 should compile without error, and you're ready for the tutorials! If you prefer to instal without the use of Cygwin, go the FAQs section for an alternate method.

4.1.5 Third-party Software

Several third-party packages help extend the capabilities of SU2, and details on their usefulness and how to obtain them are given here.

Python & Python Modules

Each of the C++ modules for SU^2 can be called directly from the command line and do not require python. However, the building of the suite and any coupling of the C++ modules, needed for design and optimization problems, can be automated by the execution of the appropriate python scripts included in the software distribution. For performing shape design, the shape_optimization.py script is available, but this script has additional dependencies on the NumPy and SciPy modules for scientific computing (including optimization routines in the SciPy library). These packages are freely available at the sites linked below.

- Python

For the shape_optimization.py script:

- NumPy
- SciPy

CGNS Library

For creating meshes around very complex geometries, third party meshing software can make the process much simpler than attempting to build meshes via scripting. With this in mind, support for the CGNS data standard has been including within SU^2. Support is currently limited to mesh file input, but additional CGNS capability will be added in future releases. The main advantage gained here is that complex meshes created in a third party software package (one that supports unstructured CGNS file export) can be used directly within SU^2. Furthermore, a converter from CGNS to the .su2 format has been built into SU^2. Users should obtain and follow the instructions supplied for building the CGNS library (Version 3.1.3 recommended) from the official CGNS site. Much more detail on compiling with and using the CGNS library for mesh input can be found on the documentation page concerning meshes.

Parallel Tools

Users wishing to run simulations on parallel architectures using domain decomposition are required to install from source and must have additional tools. The parallelization utilizes the standard Message Passing Interface (MPI), and the domain decomposition is performed using the METIS software package. When using the build script for a parallel computation, it is assumed that the top level directory for METIS (metis-4.0.3/) is in the $SU2_HOME directory.

- METIS 4.0.3
- MPI Implementation - OpenMPI or MPICH2

If you would like to install METIS and compile SU^2 using the build script, follow these directions:

1. Download METIS 4.0.3 from the link above
2. Untar the download and place the metis-4.03/ directory in the SU2v1.0/ directory (metis does not need to reside in the SU2_DDC/ directory)
3. Run the build script, which will find, compile, and link the this version of the metis library automatically

4.2 Running SU^2

Once downloaded and installed, SU^2 will be ready to run simulations and design problems. Using simple command line syntax, users can execute the individual C++ programs, specifying the problem parameters in the all-purpose configuration file. For users seeking to utilize the more advanced features of the suite (including, but not limited to, shape optimization and adaptive mesh refinement), a working python installation is required. Appropriate syntax and information for running the C++ modules and python scripts can be found below.

4.2.1 C++ Modules

As described in the SU^2 Tools page, there are seven C++ modules that are included in the distribution for SU^2. After compilation, each can be executed at the command line using a Unix-based terminal (or appropriate emulator, such as Cygwin). The executables for these modules can be found in the $SU2_HOME/¡MODULE_NAME¿/bin directories and in the $SU2_HOME/SU2Py directory. The configuration file specifies the problem and solver parameters for all SU^2 modules and must be included at runtime.

The syntax for running each C++ module separately is:

$PATH_TO_MODULE/Module_Name your_configuration_file.cfg

where Module_Name can be either SU2_CFD, SU2_GPC, SU2_MDC, SU2_MAC, SU2_DDC, SU2_GDC, SU2_PBC and your_configuration_file.cfg is the name of the configuration file for the problem. An example of a call to SU2_CFD with a configuration file "default.cfg" is included below:

./SU2_CFD default.cfg

Where the executable, SU2_CFD, and the configuration file, config.cfg, are located in the current working directory.

4.2.2 Python Scripts

The distribution of SU^2 includes several python scripts that coordinate the use of the C++ modules to perform more advanced analyses and simulations. A working installation of python is highly recommended, even for users interested in the CFD module of SU^2 only, as the compilation procedure for the source code has been automated using one of these python scripts, making the installation from source much easier. These python scripts can be found in the $SU2_HOME/SU2Py directory and are as follows:

- build_SU2.py
- continuous_adjoint.py
- finite_differences.py
- merge_solution_paraview.py
- merge_solution_tecplot.py
- merge_restart_su2.py
- divide_solution_su2.py
- mesh_adaptation.py
- parallel_computation.py
- shape_optimization.py

All of the scripts can be executed by calling python and passing the appropriate SU^2 python script and options at runtime. The syntax is as follows:

$ python script_name.py [options]

where script_name.py is the name of the script to be run, and Options is a list of options available to each script file. A brief description of the each of the scripts, their execution syntax and runtime options are included below. Users are encouraged to look at the source code for the python scripts. As with many python programs, the code is easily readable and gives the specifics of the implementation.

Analysis Scripts

Parallel Computation Script (parallel_computation.py) The parallel computation script, parallel_computation.py, coordinates the steps necessary to run SU2_CFD in parallel. The script first calls SU2_DDC, which, in turn, uses METIS to decompose the computational domain into a specified number of sub-problems. The script then calls SU2_CFD in parallel using mpirun with the indicated number of processors, sending each decomposed domain to its correspondingly-ranked MPI process. At the conclusion of the simulation, the parallel_computation.py script stitches the decomposed solutions back together by executing the merge_solution.py script, and deletes the decomposed domains and input files.

Usage: $ python parallel_computation.py [options]
Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -d DIVIDE, –divide=DIVIDE DIVIDE the numerical grid using SU2_DDC
- -o OUTPUT, –output=OUTPUT OUTPUT the domain solution

Merge Solution Scripts (merge_solution_paraview.py or merge_solution_tecplot.py) The merge solution scripts, merge_solution_paraview.py or merge_solution_tecplot.py, recombine the decomposed solution files from parallel computations into a single solution file for the corresponding visualization software (ParaView or Tecplot).

Usage: $ python merge_solution_paraview.py [options] or $ python merge_solution_tecplot.py [options]
Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -t TIMESTEP, –timestep=TIMESTEP index of the TIMESTEP
- -o OUTPUT, –output=OUTPUT OUTPUT the domain solution

Merge Restart File Script (merge_restart_su2.py) The merge restart file script, merge_restart_su2.py, recombines the decomposed internal restart files (.dat) from parallel computations into a single restart file.

Usage: $ python merge_restart_su2.py [options]
Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -t TIMESTEP, –timestep=TIMESTEP index of the TIMESTEP

Divide Solution File Script (divide_solution_su2.py) The divide solution file script, divide_solution_su2.py, divides a single internal restart file (.dat) for use on multiple processors in a parallel simulation.

Usage: $ python divide_solution_su2.py [options]
Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -t TIMESTEP, –timestep=TIMESTEP index of the TIMESTEP

Design Scripts

Continuous Adjoint Gradient Calculation (continuous_adjoint.py) The continuous adjoint calculation script, continuous_adjoint.py, automates the procedure for calculating sensitivities using the SU^2 suite using adjoint methods. The script calls SU2_CFD to first run a direct analysis to obtain a converged solution, then calls SU2_CFD again to run an adjoint analysis on the converged flow solution to obtain surface sensitivities. Perturbations in the design variables are made, then the SU2_GPC module is called to project the design variable perturbations onto the surface sensitivities calculated in the adjoint solution to arrive at the gradient of the objective function with respect to the specified design variables.

Usage: $ python continuous_adjoint.py [options]

Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -s STEP, –step=STEP finite difference STEP

Finite Difference Gradient Calculation (finite_differences.py) The finite difference calculation script, finite_difference.py, is used to calculate the gradient of an objective function with respect to specified design variables using a finite difference method. This script calls SU2_CFD repeatedly, perturbing the input design variables, and storing gradient values.

Usage: $ python finite_differences.py [options]

Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -s STEP, –step=STEP finite difference STEP

Shape Optimization Script (shape_optimization.py) The shape optimization script, shape_optimization.py, coordinates and synchronizes the steps necessary to run a shape optimization problem using the design variables and objective function specified in the configuration file. The optimization is handled using Scipy's BFGS or SLSQP optimization algorithms. Objective functions (drag, lift, etc.) are determined running a direct flow solution in SU2_CFD and gradients are obtained using the adjoint solution. For each iteration in the design process, the mesh is deformed using SU2_MDC, and the analysis is repeated until a local optimum is reached.

Usage: $ python shape_optimization.py [options]

Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -p PARTITIONS, –partitions=PARTITIONS number of PARTITIONS
- -g GRADIENT_SCALE, –gradient_scale=GRADIENT_SCALE value of the of GRADIENT_SCALE
- -c CONSTRAINTS, –constraints=CONSTRAINTS optimization with CONSTRAINTS

Grid Modification Scripts

Mesh Adaptation Script (mesh_adaptation.py) The adaptive mesh refinement script, mesh_adaptation.py, provides several methods for strategically refining simulation computational domains. Options exist for add cell density to the mesh based on flow and adjoint gradients, or based on adjoint methods for computable discretization error. Depending on the strategy selected, the script calls SU2_CFD and runs direct (and/or adjoint) simulations, appropriately modifying settings in the input configuration file. Users can specify the number of refinement cycles and cell density to add in the configuration file under the "Mesh Adaptation" section.

Usage: $ python mesh_adaptation.py [options]

Options:

- -h, –help show this help message and exit
- -f FILE, –file=FILE read config from FILE
- -o OVERWRITE_MESH, –overwrite=OVERWRITE_MESH OVERWRITE_MESH the output mesh with the adapted one

4.3 Input and Output Files

The SU^2 software suite requires several input files and produces several output files appropriate for each problem defined by the user. The configuration file defines the problem of interest, and the various settings to be used by the software suite. A mesh file contains the appropriate details defining the computational domain of interest for the problem. Optionally, a simulation may be resumed if terminated at a specified number of iterations or after achieving a specified level of convergence in the residuals, these restart files contain all the appropriate information regarding the state of the computational domain at its last time step.

4.4 Configuration file

The configuration file is a text file where users set options for the SU^2 suite. This section briefly describes the file format and other conventions. The options themselves are described at the end of this section.

The SU^2 configuration file name is of the form filename.cfg. The file extension .cfg is not optional, but the prefix can be any valid string with no spaces; e.g. config.cfg, su2-config.cfg, and flow_config.cfg are all suitable file names. The file, or a link to the file, must appear in the directory where the executables are launched. See Running SU^2 on how to supply the configuration file to either the C++ executables or the Python scripts. An example configuration file, called config_template.cfg, can be found in the $SU2_HOME directory.

The configuration file consists of only three elements:

- Options: An option in the file has the following syntax: option_name = value, where option_name is the name of the option and value is the desired option value. The value element may be a scalar data type, a list of data types, or a more complicated structure. The "=" sign must come immediately after the option_name element and is not optional. Spaces and tabs between the various elements are not significant. Lower, upper, or mixed case strings are allowed. Lists of data types can formated for appearance using commas, ()-braces, -braces, and []-braces. Some example option formats are given below.
  - mach_number = 0.8
  - FREESTREAM_VELOCITY = (5.0, 0.00, 0.00)
  - REF_ORIGIN_MOMENT= 0.25 0.0 0.0
  - KIND_TURB_MODEL = NONE
- Comments: On a given line in the file, any text appearing after a
- White space: Empty lines are ignored. On text lines that define options, white space (tabs, spaces) can be used to format the appearance of the file.

4.4.1 Option Descriptions

Problem definition options

- PHYSICAL_PROBLEM: Sets the governing equations to be solved

  Possible values: POTENTIAL_FLOW, EULER, NAVIER_STOKES, PLASMA, TWO_PHASE_FLOW, COMBUSTION

- INCOMPRESSIBLE_FORMULATION: Models the flow with an incompressibility assumption using artificial compressibility

  Possible values: YES, NO

- SHOCKTUBE_PROBLEM: Initialization as a shock tube problem

  Possible values: YES, NO

- KIND_TURB_MODEL: If PHYSICAL_PROBLE is Navier-Stokes, this sets the turbulent model

  Possible values: NONE, SA

- MATH_PROBLEM: Mathematical problem

  Possible values: DIRECT, ADJOINT, LINEARIZED, ONE_SHOT_ADJOINT

- DIVIDE_ELEMENTS: Divide elements into triangles and tetrahedrons

  Possible values: NO, YES

- RESTART_SOL: Restart solution

  Possible values: NO, YES

Free-stream quantities

- MACH_NUMBER: Free-stream Mach number (non-dimensional, based on the free-stream values)

  Possible values: must be positive

- AoA: Angle of attack in degrees

  Possible values: no restrictions

- SIDESLIP_ANGLE: Side-slip angle in degrees

  Possible values: no restrictions

- FREESTREAM_PRESSURE: Value of the free-stream pressure. Euler flows use a default value of 101325.0 N/m2

  Possible values: must be positive

- FREESTREAM_TEMPERATURE: Free-stream temperature. The default is 273.15 K.

  Possible values: must be positive

- REYNOLDS_NUMBER: Reynolds number based on the free-stream quantities (for details see Non-dimensionalization)

  Possible values: must be positive

- REYNOLDS_LENGTH: Reynolds length (in meters)

  Possible values: must be positive

- FREESTREAM_DENSITY: Free-stream density, which is used for incompressible flows only. Default values are 1.2886 Kg/m3 (air), 998.2 Kg/m3 (water).

  Possible values: must be positive

- FREESTREAM_VELOCITY: Free-stream velocity in m/s. Used for incompressible flow.

  Possible values: a list of 3 values, e.g. (5.0, 0.00, 0.00)

- FREESTREAM_VISCOSITY: Free-stream viscosity, which is used for incompressible flows only. Default values are 1.853E-5 Ns/m2 (air), 0.798E-3 Ns/m2 (water).

Fluid Properties

- GAMMA_VALUE: Ratio of specific heats, which is used for compressible flows only. Default value is 1.4 (air).

  Possible values: must be positive

- GAS_CONSTANT: Specific gas constant; used for compressible flows only. Default value is 287.87 J/kg*K (air).

  Possible values: must be positive

- PRANDTL_LAM: Laminar Prandtl number; used for compressible flows only. Default value is 0.72 (air).

  Possible values: must be positive

- PRANDTL_TURB: Turbulent Prandtl number; used for compressible flows only. Default value is 0.9 (air).

  Possible values: must be positive

- BULK_MODULUS: Value of the Bulk Modulus; used only for incompressible flows. Default values are 1.01E5 N/m^2 for air and 2.2E9 N/m^2 for water.

  Possible values: must be positive

Reference value options

- CONVERT_TO_METER: Conversion factor for converting the grid to meters.

  Possible values: must be positive

- REF_ORIGIN_MOMENT: Reference origin for moment computations.

  Possible values: a list of 3 values, e.g. (0.25, 0.00, 0.00)

- REF_LENGTH_MOMENT: Reference length for pitching, rolling, and yawing non-dimensionalization

  Possible values: must be positive

- REF_AREA: Reference area for force coefficients. A value of 0 can be used to compute the value automatically.

  Possible values: positive, or zero for automatic computation.

- REF_PRESSURE: Reference pressure. The default value is 101325.0 N/m2.

  Possible values: must be positive

- REF_TEMPERATURE: Reference temperature. The default value is 273.15 K.

  Possible values: must be positive

- REF_DENSITY: Reference density.

  Possible values: must be positive

- REF_VISCOSITY: Reference viscosity; used only for incompressible flows.

  Possible values: must be positive

- REF_VELOCITY: Reference velocity magnitude; used only for incompressible flows.

  Possible values: must be positive

Options for unsteady simulations

- UNSTEADY_SIMULATION: Type of unsteady numerical method.

  Possible values: NO, TIME_STEPPING, DUAL_TIME_STEPPING

- UNST_TIMESTEP: Time step for dual-time stepping simulations.

  Possible values: must be positive

Rotating frame options

- ROTATING_FRAME: Determines if a rotating frame is used

  Possible values: NO, YES

- ROTATIONAL_ORIGIN: Origin of the axis of rotation. Default rotation origin is (0.0, 0.0, 0.0)

  Possible values: a list of three values

- ROTATION_RATE: Angular velocity vector (rad/s). Default value is (0.0, 0.0, 0.0).

  Possible values: a list of three values

- ROT_REF_VEL: Reference speed (m/s) for computing force coefficients (e.g. tip speed)

  Possible values: must be positive

Options related to boundary conditions and surfaces

- MARKER_EULER: Boundary markers that indicate where Euler wall boundary conditions are imposed (NONE = no such boundary)

  Possible values: any set of valid surfaces found in the .su2 file

- MARKER_NS: Boundary markers that indicate where Navier-Stokes wall boundary conditions are imposed (NONE = no such boundary)

  Possible values: any set of valid surfaces found in the .su2 file

- MARKER_FAR: Boundary markers that indicate where far-field (characteristic) boundary conditions are imposed (NONE = no such boundary)

  Possible values: any set of valid surfaces found in the .su2 file

- MARKER_SYM: Boundary markers that indicate where symmetry boundaries are imposed (NONE = no such boundary)

  Possible values: any set of valid surfaces found in the .su2 file

- MARKER_INLET: Boundary markers and paramters that define inlet boundaries (NONE = no such boundary)

  Format: (marker_name, total_temperature, total_pressure, flow_angle_x, flow_angle_y, flow_angle_z)

- MARKER_OUTLET: Boundary markers and parameters that define the outlet boundaries (NONE = no such boundary)

  Format: (marker_name, total_pressure)

- MARKER_PERIODIC: Boundary markers and parameters that define periodic boundaries (NONE = no such boundary)

  Format: (marker_name, donor_marker_name, rotation_center_x, rotation_center_y, rotation_center_z, rotation_angle_x-axis, rotation_angle_y-axis, rotation_angle_z-axis, translation_x, translation_y, translation_z, ...)

- MARKER_NEARFIELD: Boundary markers and parameters that define nearfield boundaries, for equivalent area calculations (NONE = no such boundary)

- EQUIV_AREA: Determines if the equivalent area on the near-field boundary should be calculated

  Possible values: NO, YES

- EA_INT_LIMIT: Integration limits of the equivalent area

  Format: (xmin, xmax, Distance_to_NearField)

- MARKER_PLOTTING: Boundary markers that indicate which surfaces are to be plotted or optimized

  Possible values: any set of valid surfaces found in the .su2 file

- MARKER_MONITORING: Boundary markers that indicate which surfaces should be included in the functional (Cd, Cl, etc) calculation.

  Possible values: any set of valid surfaces found in the .su2 file

Options that define the numerical method

- NUM_METHOD_GRAD: Discretization used for the spatial gradients

  Possible values: GREEN_GAUSS, LEAST_SQUARES, WEIGHTED_LEAST_SQUARES

- CFL_NUMBER: Courant-Friedrichs-Lewy condition on the finest grid

  Possible values: must be positive

- CFL_RAMP: Parameters that determine how the CFL number is increased

  Format: (factor, number of iterations, CFL limit)

- ARTCOMP_FACTOR: Artifical compressibility factor; used for incompressible flows only. Default value is 5.0.

  Possible values: must be positive

- ARTCOMP_MIN: Minimum artifical compressibility; used only for incompressible flows. The default value is 0.3.

  Possible values: must be positive

- RK_ALPHA_COEFF: Runge-Kutta alpha coefficients

  Format: (alpha_1, alpha_2, alpha_3)

- RK_BETA_COEFF: Runge-Kutta beta coefficients

  Format: (beta_1, beta_2, beta_3)

- EXT_ITER: Number of maximum iterations

  Possible values: must be positive

- RES_SMOOTHING_ITER: Number of residual smoothing iterations (0 = no residual smoothing)

  Possible values: must be positive

- RES_SMOOTHING_COEFF: Smoothing factor in the residual smoothing strategy

Options related to multigrid

- FULLMG: Determines if full multigrid is to be used

  Possible values: NO, YES

- START_UP_ITER: Number of start-up iterations on the fine grid

  Possible values: must be non-negative

- MGLEVEL: Number of multi-Grid Levels (0 = no multi-grid)

  Possible values: must be non-negative

- MGCYCLE: Type of multi-grid cycle

  Possible values: 0 = V cycle, 1 = W Cycle

- MG_CFL_REDUCTION: CFL reduction factor on the coarse levels

  Possible values: between 0 and 1

- MAX_CHILDREN: Maximum number of children in the agglomeration stage

  Possible values: must be positive

- MAX_DIMENSION: Maximum length of an agglomerated element (relative to the domain)
  Possible values: must be positive
- MG_PRE_SMOOTH: Multigrid pre-smoothing iterations on each level
  Format: (iters_level_0, iters_level_1, ...)
- MG_POST_SMOOTH: Multigrid post-smoothing iterations on each level
  Format: (iters_level_0, iters_level_1, ...)
- MG_CORRECTION_SMOOTH: Number of Jacobi implicit smoothing iterations of the correction
  Format: (iters_level_0, iters_level_1, ...)
- MG_DAMP_RESTRICTION: Damping factor for the residual restriction
  Possible values: between 0 and 1
- MG_DAMP_PROLONGATION: Damping factor for the correction prolongation
  Possible values: between 0 and 1
- MG_RESTART_CYCLE: Indicates if the Multigrid cycle is restarted with the interpolated solution
  Possible values: NO, YES

Options related to the discretization

- CONV_NUM_METHOD_FLOW: Discretization scheme for the convective terms
  Possible values: JST, LAX-FRIEDRICH, ROE-1ST_ORDER, ROE-2ND_ORDER
- SLOPE_LIMITER_FLOW: Slope limiter
  Possible values: NONE, VENKATAKRISHNAN
- AD_COEFF_FLOW: 1st, 2nd and 4th difference artificial dissipation coefficients
  Format: (1st_diff_coeff, 2nd_diff_coeff, 4th_diff_coeff)
- AD_STRETCHING_FLOW: Indicates if the stretching factor for the artificial dissipation is to be used
  Possible values: NO, YES
- VISC_NUM_METHOD_FLOW: Discretization scheme for the viscous terms
  Possible values: AVG_GRAD, AVG_GRAD_CORRECTED, GALERKIN
- SOUR_NUM_METHOD_FLOW: Discretization scheme for the source term
  Possible values: PIECEWISE_CONSTANT
- TIME_DISCRE_FLOW: Time discretization scheme
  Possible values: RUNGE-KUTTA_EXPLICIT, EULER_IMPLICIT, EULER_EXPLICIT

Options related to the flow-adjoint problem

- CADJ_OBJFUNC: Objective functional associated with the dual problem
  Possible values: DRAG, LIFT, SIDEFORCE, PRESSURE, MOMENT_X, MOMENT_Y, MOMENT_Z, EFFICIENCY, EQUIVALENT_AREA, NEARFIELD_PRESSURE
- DRAG_IN_SONICBOOM: Drag penalty weight in sonic boom objective function
  Possible values: between 0 and 1
- SENS_LIMIT: Do not plot sensitivity greater than this value
- PRIMGRAD_THRESHOLD: Primitive variables gradient threshold

- CONV_NUM_METHOD_ADJ: Discretization scheme for the convective terms (continuous adjoint)
  Possible values: JST, LAX-FRIEDRICH, ROE-1ST_ORDER, ROE-2ND_ORDER
- AD_COEFF_ADJ: 1st and 4th difference artificial dissipation coefficients
  Format: (1st_diff_coeff, 4th_diff_coeff)
- AD_STRETCHING_ADJ: Indicates if the stretching factor for the artificial dissipation is to be used
  Possible values: NO, YES
- ADJ_CFL_REDUCTION: CFL reduction factor on the coarse levels
  Possible values: between 0 and 1
- VISC_NUM_METHOD_ADJ: Discretization scheme for the viscous terms (continuous adjoint)
  Possible values: DIVERGENCE_THEOREM, DIVERGENCE_THEOREM_WEISS, GALERKIN
- SOUR_NUM_METHOD_ADJ: Discretization scheme for the source term (continuous adjoint)
  Possible values: PIECEWISE_CONSTANT
- TIME_DISCRE_ADJ: Time discretization scheme
  Possible values: RUNGE-KUTTA_EXPLICIT, EULER_IMPLICIT
- FROZEN_VISC: Freeze the solution dependent part of the artificial viscosity coefficient
  Possible values: NO, YES

Options related to linearized problem

- CONV_NUM_METHOD_LIN: Discretization scheme for the convective terms
  Possible values: JST, LAX-FRIEDRICH
- AD_COEFF_LIN: 1st and 4th difference artificial dissipation coefficients
  Format: (1st_diff_coeff, 4th_diff_coeff)
- AD_STRETCHING_LIN: Indicates if the stretching factor for the artificial dissipation is to be used
  Possible values: NO, YES
- TIME_DISCRE_LIN: Time discretization scheme
  Possible values: RUNGE-KUTTA_EXPLICIT

Options related to the turbulence model

- CONV_NUM_METHOD_TURB: Discretization scheme for the convective terms
  Possible values: SCALAR_UPWIND-1ST_ORDER, SCALAR_UPWIND-2ND_ORDER
- SLOPE_LIMITER_TURB: Slope limiter
  Possible values: NONE, VENKATAKRISHNAN
- VISC_NUM_METHOD_TURB: Discretization scheme for the viscous terms
  Possible values: DIVERGENCE_THEOREM, DIVERGENCE_THEOREM_WEISS
- SOUR_NUM_METHOD_TURB: Discretization scheme for the source term
  Possible values: PIECEWISE_CONSTANT
- TIME_DISCRE_TURB: Time discretization scheme
  Possible values: EULER_IMPLICIT

Options related to the plasma model

- CONV_NUM_METHOD_PLASMA: Discretization scheme for the convective terms
  Possible values: JST, LAX-FRIEDRICH, ROE-1ST_ORDER, ROE-2ND_ORDER
- VISC_NUM_METHOD_PLASMA: Discretization scheme for the viscous terms
  Possible values: DIVERGENCE_THEOREM
- SOUR_NUM_METHOD_PLASMA: Discretization scheme for the source term
  Possible values: PIECEWISE_CONSTANT
- TIME_DISCRE_PLASMA: Time discretization scheme
  Possible values: RUNGE-KUTTA_EXPLICIT, EULER_IMPLICIT, EULER_EXPLICIT

Options related to the electric potential model

- VISC_NUM_METHOD_ELEC: Discretization scheme for the viscous terms
  Possible values: GALERKIN
- SOUR_NUM_METHOD_ELEC: Discretization scheme for the source term
  Possible values: PIECEWISE_CONSTANT

Level set options

- CONV_NUM_METHOD_LEVELSET: Discretization scheme for the convective terms
  Possible values: ROE-1ST_ORDER, ROE-2ND_ORDER
- SLOPE_LIMITER_LEVELSET: Slope limiter
  Possible values: NONE, VENKATAKRISHNAN
- TIME_DISCRE_LEVELSET: Time discretization scheme
  Possible values: RUNGE-KUTTA_EXPLICIT, EULER_IMPLICIT, EULER_EXPLICIT

Options related to the combustion model

- CONV_NUM_METHOD_COMB: Discretization scheme for the convective terms
  Possible values: ROE-1ST_ORDER
- TIME_DISCRE_COMB: Time discretization scheme
  Possible values: EULER_EXPLICIT

Options related to the turbulent-adjoint problem

- CONV_NUM_METHOD_ADJTURB: Discretization scheme for the convective terms
  Possible values: SCALAR_UPWIND-1ST_ORDER, SCALAR_UPWIND-2ND_ORDER
- SLOPE_LIMITER_ADJTURB: Slope limiter
  Possible values: NONE, VENKATAKRISHNAN
- VISC_NUM_METHOD_ADJTURB: Discretization scheme for the viscous terms
  Possible values: DIVERGENCE_THEOREM, DIVERGENCE_THEOREM_WEISS
- SOUR_NUM_METHOD_ADJTURB: Discretization scheme for the source term
  Possible values: PIECEWISE_CONSTANT
- TIME_DISCRE_ADJTURB: Time discretization scheme
  Possible values: EULER_IMPLICIT

Grid partitioning options

- NUMBER_PART: Number of partitions of the domain

  Possible values: must be non-negative

- VISUALIZE_PART: Indicates if a paraview file should be written for each partition

  Possible values: NO, YES

Grid adaptation options

- CYCLE_ADAPT: Number of adaptation cycles

  Possible values: must be non-negative

- NEW_ELEMS: Percentage of new elements, of the original number of elements, to adapt

  Possible values: between 0 and 1

- KIND_ADAPT: Type of grid adaption to use

  Possible values: NONE, FULL, FULL_FLOW, GRAD_FLOW, FULL_ADJOINT, GRAD_ADJOINT, GRAD_FLOW_ADJ, ROBUST, FULL_LINEAR, COMPUTABLE, COMPUTABLE_ROBUST, REMAINING, WAKE, HORIZONTAL_PLANE, SMOOTHING, SUPERSONIC_SHOCK, TWOPHASE

- DUALVOL_POWER: Scaling factor for the dual volume

- HORIZONTAL_PLANE_POSITION: When adapting to a horizontal plane, this indicates the position of the horizontal plane (y coord for 2D, and z coord for 3D)

- HORIZONTAL_PLANE_MARKER: Markers for the horizontal plane bondary (upper, lower)

- ANALYTICAL_SURFDEF: Indicates if an analytical definition of the geometry is to be used during adaptation.

  Possible values: NONE, NACA0012_AIRFOIL, BIPARABOLIC, NACA4412_AIRFOIL, CYLINDER

- SMOOTH_GEOMETRY: Before each computation do an implicit smoothing of the nodes coordinates

  Possible values: NO, YES

Grid deformation parameters

- DV_KIND: Type of grid deformation variables to use

  Possible values: NO_DEFORMATION, HICKS_HENNE, PARABOLIC, NACA_4DIGITS, DISPLACEMENT, ROTATION, FFD_CONTROL_POINT, FFD_DIHEDRAL_ANGLE, FFD_TWIST_ANGLE, FFD_ROTATION, FFD_CAMBER, FFD_THICKNESS, FFD_VOLUME

- DV_MARKER: Boundary markers that indicate where we are going apply the shape deformation

  Possible values: any set of valid surfaces found in the .su2 file

  - DV_PARAM: Parameters for the type of grid deformation being used
  - HICKS_HENNE format: (Lower(0)/Upper(1) side, x_Loc)
  - NACA_4DIGITS format: (1st digit, 2nd digit, 3rd and 4th digit)
  - PARABOLIC format: (1st digit, 2nd and 3rd digit)
  - DISPLACEMENT format: (x_Disp, y_Disp, z_Disp)
  - ROTATION format: (x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_CONTROL_POINT format: (Chunk, i_Ind, j_Ind, k_Ind, x_Disp, y_Disp, z_Disp)
  - FFD_DIHEDRAL_ANGLE format: (Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_TWIST_ANGLE format: (Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_ROTATION format: (Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)

- FFD_CAMBER format: (Chunk, i_Ind, j_Ind)
  - FFD_THICKNESS format: (Chunk, i_Ind, j_Ind)
  - FFD_VOLUME format: (Chunk, i_Ind, j_Ind)
  - DV_VALUE_OLD: Old value of the deformation for incremental deformations
  - DV_VALUE_NEW: New value of the shape deformation

- GRID_DEFORM_METHOD: The type of grid deformation to use

  Possible values: SPRING, TORSIONAL_SPRING, ALGEBRAIC

- GRID_DEFORM_SOLVER: The iterative solver used to solve for the grid deformation (if applicable)

  Possible values: SYM_GAUSS_SEIDEL, CONJUGATE_GRADIENT, PREC_CONJUGATE_GRADIENT, FLEXIBLE_GMRES

- GRID_DEFORM_ERROR: Tolerance used in the solution of the grid deformation linear system

  Possible values: must be non-negative

- GRID_DEFORM_BOX: Move points that are inside the FFD box only

  Possible values: NO, YES

Options related to measuring convergence

- CONV_CRITERIA: Convergence criteria to use

  Possible values: CAUCHY, RESIDUAL

- RESIDUAL_REDUCTION: Residual reduction (order of magnitude) with respect to the maximum residual

  Possible values: must be positive

- RESIDUAL_MINVAL: Minimum absolute tolerance for the residual (log10 of the residual)

  Possible values: must be negative

- STARTCONV_ITER: The iteration at which monitoring convergence begins

- CAUCHY_ELEMS: Number of iterations used to calculate the Cauchy criterion

  Possible values: must be positive

- CAUCHY_EPS: Tolerance used to determine convergence for the Cauchy criterion

  Possible values: between 0 and 1

- CAUCHY_FUNC_FLOW: Functional used in the Cauchy convergence criteriion for the flow

  Possible values: LIFT, DRAG, NEARFIELD_PRESS, SENS_GEOMETRY, SENS_MACH, DELTA_LIFT, DELTA_DRAG

- CAUCHY_FUNC_ADJ: Functional used in the Cauchy convergence criteriion for the adjoint problem

  Possible values: LIFT, DRAG, NEARFIELD_PRESS, SENS_GEOMETRY, SENS_MACH, DELTA_LIFT, DELTA_DRAG

- CAUCHY_FUNC_LIN: Functional used in the Cauchy convergence criteriion for the linearized problem

  Possible values: LIFT, DRAG, NEARFIELD_PRESS, SENS_GEOMETRY, SENS_MACH, DELTA_LIFT, DELTA_DRAG

- ONESHOT_CAUCHY_EPS: Tolerance used to measure one-shot convergence

  Possible values: between 0 and 1

- FULLMG_CAUCHY_EPS: Epsilon for full multigrid method evaluation

  Possible values: between 0 and 1

Input/Output options

- MESH_FILENAME: Mesh input file name
- MESH_FORMAT: Mesh input file format
  Possible values: SU2, CGNS, NETCDF_ASCII
- CGNS_TO_DPL: Convert a CGNS mesh to SU^2 format
  Possible values: YES, NO
- NEW_DPL_FILENAME: Name of the converted SU^2 mesh file
- MESH_OUT_FILENAME: Mesh output file name
- SOLUTION_FLOW_FILENAME: Name of file containing flow restart data (to be read)
- SOLUTION_LIN_FILENAME: Name of file containing linearized problem restart data (to be read)
- SOLUTION_ADJ_FILENAME: Name of file containing adjoint problem restart data (to be read)
- OUTPUT_FORMAT: Output file format
  Possible values: PARAVIEW, TECPLOT
- CONV_FILENAME: File name for the convergence history (w/o extension)
- RESIDUAL_FILENAME: File name for the residual history (.vtk extension)
- RESTART_FLOW_FILENAME: Name of file where flow restart data is written
- RESTART_ADJ_FILENAME: Name of file where adjoint-problem restart data is written
- RESTART_LIN_FILENAME: Name of file where linearized-problem restart data is written
- VOLUME_FLOW_FILENAME: Output file name (w/o extension) for the flow variables
- VOLUME_ADJ_FILENAME: Output file name (w/o extension) for the adjoint variables
- VOLUME_LIN_FILENAME: Output file name (w/o extension) for the linearized-problem variables
- GRAD_FILENAME: Output file name for the gradient of the conservative variables (with .vtk extension)
- GRAD_OBJFUNC_FILENAME: Output file name for the objective function gradient (with .dat extension)
- SURFACE_FLOW_FILENAME: Output file name (w/o extension) for the flow variables on the surface
- SURFACE_ADJ_FILENAME: Output file name (w/o extension) for the adjoint variables on the surface
- SURFACE_LIN_FILENAME: Output file name (w/o extension) for the linearized-problem variables on the surface
- WRT_SOL_FREQ: Frequency for writing solution file
  Possible values: non-negative integer
- WRT_CON_FREQ: Frequency for writing the convergence history
  Possible values: non-negative integer

Options related to optimal shape design

- OBJFUNC: Targent objective functional to optimize

  Possible values: DRAG, LIFT, SIDEFORCE, PRESSURE, MOMENT_X, MOMENT_Y, MOMENT_Z, EFFICIENCY, EQUIVALENT_AREA, NEARFIELD_PRESSURE

- CONSTRAINT: Indicates if a constraint functional is present

  Possible values: NONE, LIFT

- MIN_LIFT: Minimum value for the lift coefficient
- MAX_DRAG: Maximum value for the drag coefficient
- MIN_PITCH: Minimum value for the pitching moment coefficient
- MAX_PITCH: Maximum value for the pitching moment coefficient
- DEFINITION_DV: List of design variables. Each set of design variables is separated by semicolons.

  General format: (DV_SET_1) ; (DV_SET_2) ; ...

  Specific formats:

  - HICKS_HENNE format: (1, Scale — Mark. List — Lower(0)/Upper(1) side, x_Loc)
  - NACA_4DIGITS format: (4, Scale — Mark. List — 1st digit, 2nd digit, 3rd and 4th digit)
  - DISPLACEMENT format: (5, Scale — Mark. List — x_Disp, y_Disp, z_Disp)
  - ROTATION format: (6, Scale — Mark. List — x_Axis, y_Axis, z_Axis, x_Turn, y_Turn, z_Turn)
  - FFD_CONTROL_POINT format: (7, Scale — Mark. List — Chunk, i_Ind, j_Ind, k_Ind, x_Mov, y_Mov, z_Mov)
  - FFD_DIHEDRAL_ANGLE format: (8, Scale — Mark. List — Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_TWIST_ANGLE format: (9, Scale — Mark. List — Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_ROTATION format: (10, Scale — Mark. List — Chunk, x_Orig, y_Orig, z_Orig, x_End, y_End, z_End)
  - FFD_CAMBER format: (11, Scale — Mark. List — Chunk, i_Ind, j_Ind)
  - FFD_THICKNESS format: (12, Scale — Mark. List — Chunk, i_Ind, j_Ind)
  - FFD_VOLUME format: (13, Scale — Mark. List — Chunk, i_Ind, j_Ind)
  - MACH_NUMBER format: (101, Scale — Markers List)
  - AOA format: (102, Scale — Markers List)

  Example:

  DEFINITION_DV = (1, 0.001 — AIRFOIL — 0, 0.1); (1, 0.001 — AIRFOIL — 0, 0.2);

4.5 Mesh files

SU^2 mainly uses a native mesh file format as input into the various suite components. Limited support for the CGNS data format has also been included as an input mesh format. CGNS support can be useful when it is necessary to create complex geometries in a third party mesh generation package that can export CGNS files. A converter from CGNS to the native format is built in to SU^2. Details on how to create and use these mesh formats is given below.

4.5.1 CGNS Format

The native format is straightforward and readable, and meshes for simple or analytic geometries can be made very easily using scripts like those provided below. However, for creating meshes around very complex geometries, third party meshing software can make the process much simpler than attempting to build meshes via scripting. With this in mind, support for the CGNS data standard has been included within SU^2. Support is currently limited to mesh file input, but additional CGNS capability will be added in future releases. The main advantage gained here is that complex meshes created in a third party software package (one that supports unstructured CGNS file export) can be used directly within SU^2. Furthermore, a converter from CGNS to the .su2 format has been built into SU^2. While serial simulations with input CGNS meshes will execute in SU2_CFD, based upon the current level of CGNS support in SU^2 (currently no parallel support or solution output in CGNS format, for example), it is recommended that the user converts their meshes to the .su2 format, in general.

Compiling with CGNS Support

First, obtain and follow the instructions supplied for building the CGNS library (Version 3.1.3 recommended) on the machine where SU^2 will be running. The resulting library will be linked during the build process of SU2_CFD which is the only module that officially supports CGNS meshes at the moment. There are two methods for compiling with CGNS support: using the build_SU2.py script with special options, or by directly manipulating the makefiles.

To use the python build script to compile with CGNS support, the user needs to include the "–with-cgns" option followed by the paths to the CGNS library and header file. The options for the build script can always be viewed by typing "python build_SU2.py -h" at the command line. A typical call to the build script on Mac OS X for compiling with CGNS support might look like the following:

```
~/SU2/SU2Py $ python build_SU2.py -o macosx –with-cgns
–cgns-inc-path=-I/usr/local/include –cgns-lib-path=usr/local/lib/libcgns.a
```

By default, it is assumed that the CGNS library and header file can be found in /usr/local/lib/ and /usr/local/include/, respectively. If your CGNS installation is in the default location, the "–cgns-inc-path" and "–cgns-lib-path" flags are optional. Otherwise, supply the paths to these files on your machine.

As mentioned, CGNS support can also be included by direct manipulation of the makefiles. A typical makefile.in for your machine might look like the following (found in the SU2_CFD/config/ directory):

```
CXX = g++
CXXFLAGS = -O3 -Wall -frerun-cse-after-loop -fomit-frame-pointer \
-fexpensive-optimizations -Wno-non-virtual-dtor -DNO_METIS \
-DNO_MPI -DNO_OMP -DNO_CGNS
ifeq (,$(findstring -DNO_CGNS,$(CXXFLAGS)))
INC_CGNS = -I/usr/local/include
LIB_CGNS = -L/usr/local/lib -lcgns
endif
```

To compile with CGNS support, first remove the "-DNO_CGNS" options from the list of C++ compiler flags. If this flag does not appear, the conditional statement below will activate, and the paths to both the CGNS library and header file must be supplied in the "LIB_CGNS" and "INC_CGNS" variables, respectively. Again, it is assumed that the CGNS library was built and installed in /usr/local/lib/ under the name libcgns.a and that the header file, cgnslib.h can be found in usr/local/include/. If you have built the library elsewhere, simply adjust the two paths to match their location on your machine. Note that if you are not building with CGNS support, these paths are ignored completely.

Using and Converting CGNS Meshes

In order to use a CGNS mesh (assuming the CGNS library has been installed and SU^2 successfully compiled), the user simply needs to specify the input mesh format to be "CGNS" in the configuration file for their simulation. The configuration file option is "MESH_FORMAT=" and appears as:

```
%
% Mesh input file format (SU2, CGNS, NETCDF_ASCII)
MESH_FORMAT= CGNS
```

Furthermore, a CGNS mesh can be converted to the .su2 format by setting the "CONVERT_TO_SU2=" flag to "YES" and supplying a new filename for the converted mesh in the "NEW_SU2_FILENAME=" option. These options might appear as follows in the configuration file:

```
%
% Convert a CGNS mesh to SU2 format (YES, NO)
CGNS_TO_SU2= YES
%
% Converted SU2 mesh filename
NEW_SU2_FILENAME= new_mesh.su2
```

It is important to note that SU^2 will not use any specific boundary conditions which are set in the meshing software package before exporting the CGNS mesh. However, it will use the names given to each boundary as the marker tags. These marker tags are used to set the boundary conditions in the configuration file. Therefore, it is recommended that the user give names to each boundary in their mesh generation package before exporting to CGNS. If you do not know the number of markers or their tags within a CGNS file, you can simply attempt a simulation in SU2_CFD (leaving out the boundary information in the configuration file at first), and during the preprocessing stage, SU^2 will read and print the names of all boundary markers to the console along with other grid information before throwing an error for incomplete boundary definitions. The user can then incorporate these marker tags into the configuration file with the appropriate boundary conditions.

4.5.2 Native Format (.su2)

In keeping with the open source nature of the project, SU^2 relies mostly on its own native mesh format. The format is meant to be simple and readable. A description of the mesh and some examples are below.

Description

The SU^2 mesh format carries an extension of .su2, and the files are in a readable ASCII format. As an unstructured code, SU^2 requires information about both the node locations as well as their connectivity. The connectivity description provides information about the types of elements (triangle, rectangle, tetrahedron, hexahedral, etc.) that make up the volumes in the mesh and also which nodes make up each of those elements. Lastly, the boundaries of the mesh, or markers, are given names, or tags, and their connectivity is specified in a similar manner as the interior nodes.

Specification

Consider the following simple, 2-D mesh for a square domain consisting of 8 triangular elements. It will be used to explain the .su2 mesh format.

The first line of the .su2 mesh declares the dimensionality of the problem. SU^2 can handle 2-D or 3-D geometries. As a note, for 2-D simulations, it is recommended that a truly 2-D mesh is used (no z-coordinates) rather than a quasi-2-D mesh (one or more cells deep in the third dimension with symmetry boundary conditions). For the 2-D square, the dimension is defined as follows:

```
%
% Problem dimension
%
NDIME= 2
```

SU^2 searches specifically for the keyword "NDIME=" in order to set the dimension, and the dimension value will be stored for use throughout the code. This value would be 3 for a 3-D mesh. Note that while "

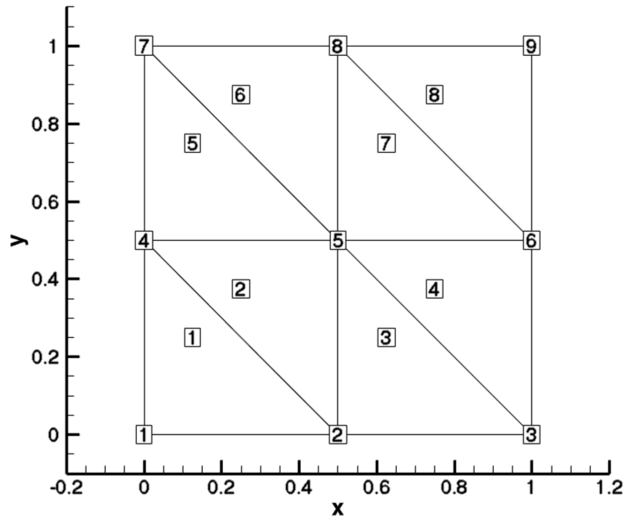


Figure 4.1: 2-D mesh example.

```
%
% Inner element connectivity
%
NELEM= 8
5 0 1 3 0
5 1 4 3 1
5 1 2 4 2
5 2 5 4 3
5 3 4 6 4
5 4 7 6 5
5 4 5 7 6
5 5 8 7 7
```

SU^2 is based on unstructured mesh technology, and thus supports several element types for both 2-D and 3-D elements. Unlike for structured meshes where a logical, ordered indexing can be assumed for neighboring nodes and their corresponding cells (rectangles in 2-D and hexahedral elements in 3-D), for an unstructured mesh, a list of nodes that make up each element, or the connectivity as it is often called, must be provided. First, SU^2 will search for the string "NELEM=" and then store the number of interior elements. This value is given first, as it is used to set up a loop over all of the elements which must immediately follow this line. For the square mesh above, this corresponds to the 8 triangular interior elements which are labeled from 1 to 8.

Each following line describes the connectivity of a single element. The first integer on each line is a unique identifier for the type of element that is described. SU^2 supports line, triangle, rectangle, tetrahedral, pyramid, wedge, and hexahedral elements. The identifiers follow the VTK format:

Element Type	Identifier
Line	3
Triangle	5
Rectangle	9
Tetrahedral	10
Hexahedral	12
Wedge	13
Pyramid	14

In our square mesh, all elements are triangles, and thus the identifier (first integer) on all lines is 5. Following the identifier is a list of the node indices that make up the element. Each triangular element will have 3 nodes specified, a rectangular element would have 4 nodes specified, a tetrahedral element would have 4 nodes specified, and so on. The final value is the element index given to each interior element in the mesh. Note that the .su2 format indexes the nodes and elements starting from zero, as opposed to starting from 1 as Tecplot does, which was used to create the mesh image. For example, take the triangular element described in the first line which is indexed as 0 (1 in Tecplot). The SU^2 nodes are given as (0,1,3) which would correspond to (1,2,4) in Tecplot. Looking at the figure of the mesh above, we see that this is the lower left triangular element. The ordering of the nodes given in the connectivity list for a specific element is important, and the user is referred to the VTK format guide for the correct ordering for each supported element type (page 9).

After the connectivity information for all interior elements, the coordinates for each of the nodes are given. This is specified in the .su2 format as:

```
%
% Node coordinates
%
NPOIN= 9
0.00000000000000 0.00000000000000 0
0.50000000000000 0.00000000000000 1
1.00000000000000 0.00000000000000 2
0.00000000000000 0.50000000000000 3
0.50000000000000 0.50000000000000 4
1.00000000000000 0.50000000000000 5
0.00000000000000 1.00000000000000 6
0.50000000000000 1.00000000000000 7
1.00000000000000 1.00000000000000 8
```

Again, SU^2 searches for the string "NPOIN=" and stores the total number of nodes in the mesh. In this case, there are 9 nodes in the 3x3 square above. Immediately after the node number specification comes the list of node coordinates in cartesian (x,y,z) space. Each line gives the coordinates for a single node followed by its index number. The node index numbers are the indices used for specifying the connectivity information for elements. For a 2-D mesh, the x and y coordinates are given followed by the index, but a 3-D mesh would give x, y, and z, followed by the index. The location of each node can be confirmed in space by comparing with the figure above after adding 1 to the index value.

The final component of the mesh is a description of all boundaries (which we call markers), including a name (what we call a tag). For each boundary, the connectivity information is given which is based off of the same node indices given above. For a 2-D mesh, only line elements are supported along the boundaries. For a 3-D mesh, triangular and rectangular elements are the possible options for boundary elements. This section of the .su2 mesh file appears as:

```
%
% Boundary elements
%
NMARK= 4
MARKER_TAG= lower
MARKER_ELEMS= 2
3 0 1
3 1 2
MARKER_TAG= right
MARKER_ELEMS= 2
3 2 5
3 5 8
MARKER_TAG= upper
MARKER_ELEMS= 2
3 8 7
3 7 6
MARKER_TAG= left
MARKER_ELEMS= 2
3 6 3
3 3 0
```

First, the number of boundaries, or markers, is specified using the "NMARK=" string. Then for each marker, a name, or tag, is specified using "MARKER_TAG=." This tag can be any string name, and the tag name is used in the configuration file for the solver when specifying boundary conditions. Here, the tags "lower," "right," "upper," and "left" would be used. The number of elements on each marker, using "MARKER_ELEMS=," must then be specified before listing the connectivity information as is done for the interior mesh elements at the start of the file. Again, the unique VTK identifier is given at the start of each line followed by the node list for that element. Note that no index is given for boundary elements. For our example, only line elements (identifier 3) exist along the markers, and on each boundary of the square there are 2 edges of a triangle that make up the marker. These elements can again be verified with the mesh figure above.

4.5.3 Examples

Attached here is the simple square mesh from above in .su2 format, along with codes for creating this file in the Python, C++, and Fortran 90 programming languages. These scripts are meant to be examples of how to write .su2 meshes in a few common languages which can be easily modified for creating new meshes:

Square mesh: square.su2 Python square mesh generator: square.py C++ square mesh generator: square.cpp Fortran 90 square mesh generator: square.f90

4.6 Solution and Restart files

SU^2 is capable of outputting solution files that can be visualized in either ParaView (.vtk) or Tecplot (.plt). Information on obtaining these packages can be found on the installation page. At the end of each simulation, SU^2 will output several files that contain all of the necessary information for visualization and restart. For a direct flow solution, these files might look like the following:

flow.plt or flow.vtk - full volume flow solution. surface_flow.plt or surface_flow.vtk - flow solution along the airfoil surface. surface_flow.csv - comma separated values (.csv) file containing values along the airfoil surface. restart_flow.dat - restart file in an internal format for restarting this simulation in SU^2. history.plt or history.csv - file containing the convergence history information. restart_flow.dat - readable restart file which contains the flow solution at each node (description below). For adjoint solutions, the file names are slightly different:

adjoint.plt or adjoint.vtk - full volume adjoint solution. surface_adjoint.plt or surface_adjoint.vtk - adjoint solution along the airfoil surface. surface_adjoint.csv - comma separated values (.csv) file containing values along the airfoil surface. restart_adj_cd.dat - restart file in an internal format for restarting this simulation in SU^2. Note that the name of the objective appears in the file name. history.plt or history.csv - file containing the convergence history information. restart_adj_cd.dat - readable restart file which contains the adjoint solution at each node (description below) for this objective. SU^2 uses a simple and readable format for the restart files. The SU^2 solution format has the extension .dat, and the files are in readable ASCII format. The solution (conservative variables) is provided at each vertex of the numerical grid, and no information about the connectivity or coordinates is included in the file. For the sake of clarity, the vertex index is provided at the beginning of each line, though this index value is not used by the code, and the sorting of the points is (and must be kept) the same as in the mesh file. The restart files are used to restart the code from a previous solution and also to run the adjoint simulations, which need a flow solution. In order to run an adjoint simulation, the user must first change the name of the restart_flow.dat file to solution_flow.dat in the execution directory. It is important to note that the adjoint solver will create a different restart file for each objective function.

4.7 Non-dimensionalization

The non-dimensionalization of the RANS equations in SU^2 is based on the document "Non-dimensionalization of the Navier-Stokes Equations" written by Prof. Feng Liu at the Department of Mechanical and Aerospace Engineering at Stanford University. The implemented non-dimensionalization satisfies the following objectives:

- Equations maintain the same form independent of the use of units.
- Proper identification of similarity parameters and similar solutions.
- Easy interpretation of numerical values.
- Minimize the number of coefficients in equations to reduce computation.
- Normalized quantities be of order 1 if possible.

The following table shows the references values that SU^2 is using. The variables are divided in 4 big groups:

- Basic independent variables of which the reference values can be arbitrarily chosen.
- Derived variables whose reference values are determined from the choice of the basic
- independent variables in group 1.
- Dimensionless parameters based on the reference variables.
- Non-dimensional coefficients.

	Variables	General	SU2	SI
1a	length	lref	lref (input)	m
	pressure	pref	pref (input)	N/m2
	density	ref	ref (input)	kg/m3
	temperature	Tref	Tref (input)	K

	Variables	General	SU2	SI
1b	velocity	uref	(pref/ref)	m/s
	time	tref	lref/uref	s
	dynamic viscosity	ref	refureflref	kg/(m.s)
	rotor speed	ref	uref/lref	1/s
	external body force	bref	u2ref/lref	m/s2

	Variables	General	SU2	SI
2b	kinematic viscosity	ref=ref/ref		m2/s
	strain rate	Sref=uref/lref		1/s
	stress	ref=pref		N/m2
	specific energy	eref=u2ref		J/kg
	specific enthalpy	href=eref		J/kg
	specific entropy	sref=eref/Tref		J/(kg.K)
	heat flux	qref=referefuref		J/(m2.s)
	gas constant	Rref=eref/Tref		J/(kg.K)
	heat capacity	cpref=Rref		J/(kg.K)
	heat capacity	cvref=Rref		J/(kg.K)
	heat conductivity	kref=cprefref		W/(K.m)

	Variables	General	SU2	SI
2c	turbulent kinetic energy	kref=u2ref		J/kg
	turbulent specific dissipation	ref=uref/lref		1/s

	Variables	General	SU2	SI
3	ratio of specific heats	$\gamma = C_p/C_v$	input	
	dimensionless gas constant	R=R*/Rref	input R*	
	molecular Prandtl number	Prl=Cp/k	input	
	turbulent Prandtl number	Prt=Cpt/kt	input	

	Variables	General	SU2	SI
4	Strouhal number	(St)ref=lref/(uref.tref)	1	1
	Euler number	(Eu)ref=pref/(refu2ref)	1	1
	Reynolds number	(Re)ref=refureflref/ref	1	1
	Rossby number	(Ro)ref=uref/(reflref)	1	1
	Froude number	(Fr)ref=uref/(breflref)	1	1

Chapter 5

User's Tutorials

Instead of writing a very detailed user manual, the approach has been taken to teach the various aspects of the SU^2 code through a range of tutorials. These are numbered roughly in order of their complexity and how experienced with the code the user may need to be, noting that the more advanced tutorials may assume the user has already worked through the earlier ones. Each tutorial attempts to present new features of SU^2 and contains explanations for the key configuration file options. For more information on the exact learning goals of a tutorial, these can be seen at the beginning of each.

5.1 Tutorial 1 - Bump in a Channel

5.1.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of internal, inviscid flow through a 2-D geometry. The specific geometry chosen for the tutorial is a channel with a bump along the lower wall. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 2-D Euler equations
- Multigrid
- JST numerical scheme in space
- Euler implicit time integration
- Inlet, Outlet, and Euler Wall boundary conditions

The intent of this tutorial is to introduce a simple, inviscid flow problem and to explain how boundary markers are used within SU^2. This tutorial is especially useful for showing how an internal flow computation can be performed using the inlet and outlet boundary conditions.

5.1.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/inv_CHANNEL/ directory. You will need the configuration file (inv_channel.cfg) and the mesh file (mesh_channel_65x33.su2).

5.1.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow through the channel using SU^2. It is assumed you have already obtained and compiled the SU2_CFD. If you have yet to complete these requirements, please see the Download and Installation pages.

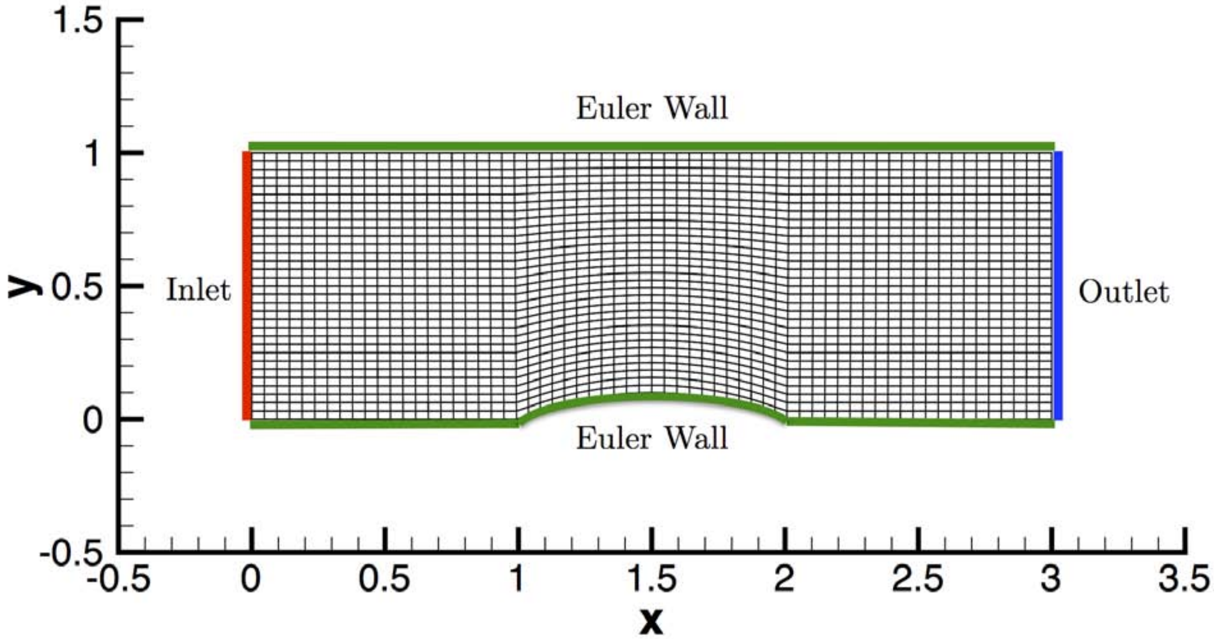


Figure 5.1: The computational mesh with boundary conditions highlighted.

5.1.4 Background

This example uses a 2-D channel geometry which features a circular bump along the lower wall. It is meant to be a simple test in inviscid flow for the subsonic inlet and outlet boundary conditions that are required for an internal flow calculation. The geometry is based on the an example in Chapter 11 of Numerical Computation of Internal and External Flows: The Fundamentals of Computational Fluid Dynamics (Second Edition) by Charles Hirsch.

5.1.5 Problem Setup

This problem will solve the for the flow through the channel with these conditions:

- Inlet Stagnation Temperature = 288.6 K
- Inlet Stagnation Pressure = 102010.0 N/m^2
- Inlet Flow Direction, unit vector (x,y,z) = (1.0, 0.0, 0.0)
- Outlet Static Pressure = 101300.0 $N/m2$
- Resulting Mach number = 0.1

Mesh Description

The channel is of length 3L, height L, with a circular bump centered along the lower wall with height 0.1L. For the SU^2 mesh, L = 1.0 was chosen, as seen in the figure of the mesh below. The mesh is structured (rectangles) with 65 nodes along the length of the channel and 33 nodes along the height. Two other, finer meshes are also included in the test case folder (mesh_channel_129x65.su2, mesh_channel_257x129.su2), if you are interested in performing any grid comparison studies. The file mesh_channel_65x33.SU^2 contains the following mesh:

The boundary conditions for the channel are also highlighted in the figure. Inlet, outlet, and Euler wall boundary conditions are used. The Euler wall boundary condition enforces flow tangency at the upper and lower walls. It is important to note that the subsonic inlet and outlet boundary conditions are based on characteristic information, meaning that only certain flow quantities can be specified at the inlet and outlet.

In SU2, the stagnation temperature, stagnation pressure, and a unit vector describing the incoming flow direction must all be specified. At an exit boundary, only the static pressure is required. These options are explained in further detail below under configuration file options. If there are multiple inlet boundaries for a problem, this information must be specified for each boundary.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Here we explain some details on markers and boundary conditions:

```
%
% Euler wall boundary marker(s) (NONE = no marker)
MARKER_EULER= ( upper_wall, lower_wall )
%
% Inlet boundary marker(s) (NONE = no marker)
% Format: ( inlet marker, total temperature, total pressure,
% flow_direction_x, flow_direction_y, flow_direction_z, ... )
% where flow_direction is a unit vector.
MARKER_INLET= ( inlet, 288.6, 102010.0, 1.0, 0.0, 0.0 )
%
% Outlet boundary marker(s) (NONE = no marker)
% Format: ( outlet marker, back pressure (static), ... )
MARKER_OUTLET= ( outlet, 101300.0 )
%
% Marker(s) of the surface to be plotted or designed
MARKER_PLOTTING= ( lower_wall )
%
% Marker(s) of the surface where the functional (Cd, Cl, etc.)
% will be evaluated
MARKER_MONITORING= ( upper_wall, lower_wall )
```

The 4 different boundary markers (upper_wall, lower_wall, inlet, and outlet) are each given a specific type of boundary condition. For the inlet and outlet boundary conditions, the additional flow conditions are specified directly within the configuration option. The format for the inlet boundary condition is (marker name, inlet stagnation pressure, inlet stagnation pressure, x-component of flow direction, y-component of flow direction, z-component of flow direction) where the final three components make up a unit flow direction vector (magnitude = 1.0). In this problem, the flow is exactly aligned with the x-direction of the coordinate system, and thus the flow direction vector is (1.0, 0.0, 0.0). The outlet boundary format is (marker name, exit static pressure). Any boundary markers that are listed in the MARKER_PLOTTING option will be written into the surface solution file. Any surfaces on which an objective such as Cl or Cd is to be calculated must be included in the MARKER_MONITORING option.

Some integration options:

```
%
% Time discretization (RUNGE-KUTTA_EXPLICIT, EULER_IMPLICIT,
% EULER_EXPLICIT)
TIME_DISCRE_FLOW= EULER_IMPLICIT
%
% Courant-Friedrichs-Lewy condition of the finest grid
CFL_NUMBER= 8.0
%
% Multi-Grid Levels (0 = no multi-grid)
MGLEVEL= 3
```

The simplicity of this problem and mesh permits an aggressive CFL number of 8 for the Euler Implicit time integration. Convergence is also accelerated with three levels of multigrid.

Setting the convergence criteria:

```
%
% Convergence criteria (CAUCHY, RESIDUAL)
CONV_CRITERIA= RESIDUAL
%
% Residual reduction (order of magnitude with respect to the initial value)
RESIDUAL_REDUCTION= 6
%
% Min value of the residual (log10 of the residual)
RESIDUAL_MINVAL= -12
%
% Start convergence criteria at iteration number
STARTCONV_ITER= 10
```

There are three different types of criteria for terminating a simulation in $SU^2$2: running a specified number of iterations (EXT_ITER option), reducing the density residual by a specified order of magnitude, or by converging an objective, such as drag, to a certain tolerance. The most common convergence criteria is the RESIDUAL option which is used in this tutorial by setting the CONV_CRITERIA. The RESIDUAL_REDUCTION option controls how many orders of magnitude reduction in the density residual are required for convergence, and RESIDUAL_MINVAL sets the minimum value that the residual is allowed to reach before automatically terminating. Because the residuals often increase slightly at the beginning of a simulation before decreasing, the user can set a specific iteration number to use for the initial value of the density residual using the STARTCONV_ITER option. For example, the simulation for the inviscid channel will terminate once the density residual reaches a value that is 6 orders of magnitude smaller than its value at iteration 10.

5.1.6 Running SU^2

The channel simulation for the 65x33 node mesh is small and will execute quickly on a single workstation or laptop, and this case will be run in a serial fashion. To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
2. Copy the config file (inv_channel.cfg) and the mesh file (mesh_channel_65x33.su2) to this directory.
3. Run the executable by entering "./SU2_CFD inv_channel.cfg" at the command line.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria.
5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.1.7 Results

The following images show some SU^2 results for the inviscid channel problem.

5.2 Tutorial 2 - Inviscid ONERA M6

5.2.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, inviscid flow around a 3-D geometry. The specific geometry chosen for the tutorial is the classic ONERA M6 wing. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 3-D Euler equations
- Multigrid

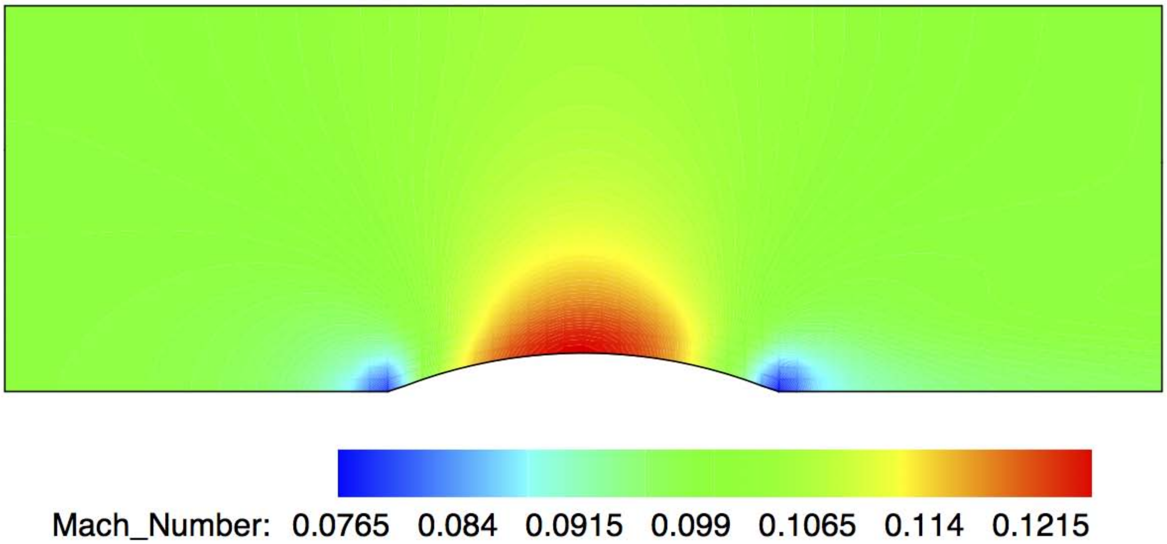


Figure 5.2: Mach number contours for the 2-D channel.

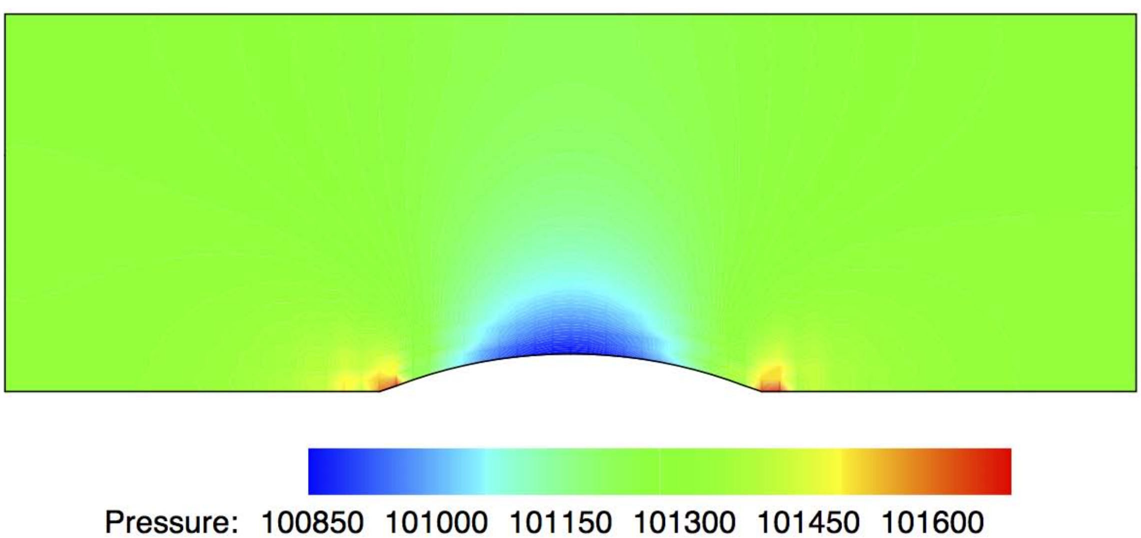


Figure 5.3: Pressure contours for the 2-D channel.

- JST numerical scheme in space
- Euler implicit time integration
- Euler Wall, Symmetry, and Farfield boundary conditions
- Code parallelism (recommended)

We will also discuss the details for setting up 3-D flow conditions and some of the multigrid options within the configuration file.

5.2.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/inv_ONERAM6/ directory. You will need the configuration file (inv_ONERAM6.cfg) and the mesh file (mesh_ONERAM6_inv.su2).

5.2.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow around the ONERA M6 using SU^2. The tutorial will also address procedures for both serial and parallel computations. It is assumed that you have already obtained and compiled the SU2_CFD code for a serial computation or both the SU2_CFD and SU2_DDC codes for a parallel computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.2.4 Background

This test case is for the ONERA M6 wing in inviscid flow. The ONERA M6 wing was designed in 1972 by the ONERA Aerodynamics Department as an experimental geometry for studying three-dimensional, high Reynolds number flows with some complex flow phenomena (transonic shocks, shock-boundary layer interaction, separated flow). It has become a classic validation case for CFD codes due to the simple geometry, complicated flow physics, and availability of experimental data. This test case will be performed in inviscid flow at a transonic Mach number.

5.2.5 Problem Setup

This problem will solve the for the flow past the wing with these conditions:

- Freestream Pressure = 101325.0 N/m2
- Freestream Temperature = 273.15 K
- Freestream Mach number = 0.8395
- Angle of attack (AoA) = 3.06 deg

These transonic flow conditions will cause the typical "lambda" shock along the upper surface of the lifting wing.

Mesh Description

The computational domain is a large parallelepiped with the wing half-span on one boundary in the x-z plane. The mesh consists of 582,752 tetrahedral elements and 108,396 nodes. Three boundary conditions are employed: Euler wall on the wing surface, the far-field characteristic-based condition on the far-field markers, and a symmetry boundary condition for the marker where the wing half-span is attached. The symmetry condition acts to mirror the flow about the x-z plane, reducing the complexity of the mesh and the computational cost. Images of the entire domain and the triangular elements on the wing surface are shown below.

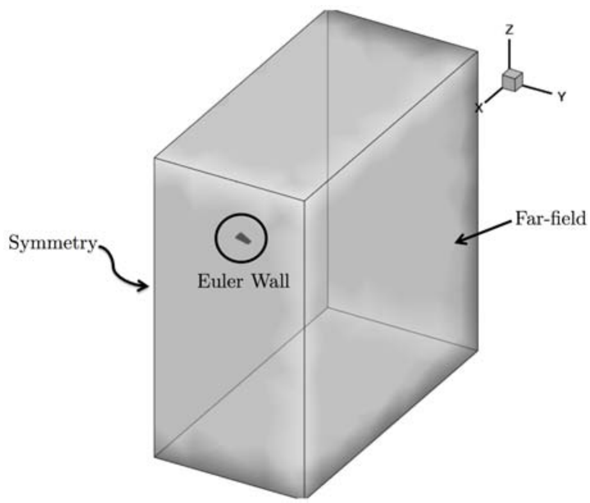


Figure 5.4: Far-field view of the computational mesh with boundary conditions.

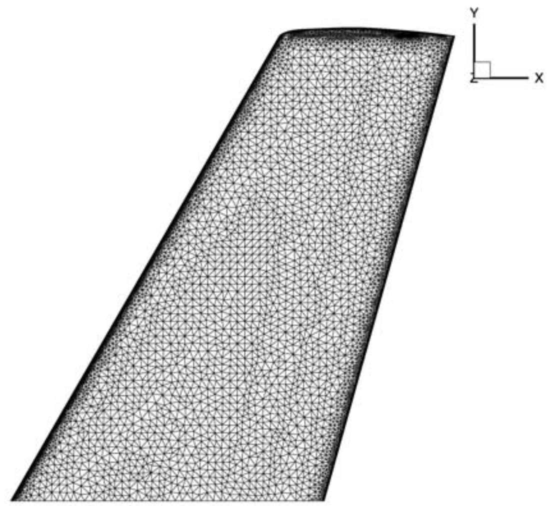


Figure 5.5: Close-up view of the unstructured mesh on the top surface of the ONERA M6 wing.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Setting up 3-D flow conditions:

```
%
% Mach number (non-dimensional, based on the free-stream values)
MACH_NUMBER= 0.8395
%
% Angle of attack (degrees)
AoA= 3.06
%
% Side-slip angle (degrees)
SIDESLIP_ANGLE= 0.0
%
% Free-stream pressure (101325.0 N/m2 by default, only for Euler
% equations)
FREESTREAM_PRESSURE= 101325.0
%
% Free-stream temperature (273.15K by default)
FREESTREAM_TEMPERATURE= 273.15
```

For an inviscid problem such as this, the flow conditions are completely defined by an input Mach number, flow direction, freestream pressure, and freestream temperature. The input Mach number is transonic at 0.8395. The freestream temperature and pressure have been set to standard sea level values for air at 101325.0 N/m2 and 273.15 K, respectively. The flow field will be initialized to these freestream values everywhere in the domain. Lastly, it is very important to note the definition of the freestream flow direction in 3-D. The default freestream direction (AoA = 0.0 degrees and SIDESLIP_ANGLE = 0.0 degrees) is along the positive x-axis without any components in the y- or z-directions. Referring to Figure (1), we see that AoA = 3.06 degrees will result in a non-zero freestream velocity in the positive z-direction. While zero for this problem, setting the SIDESLIP_ANGLE to a non-zero value would result in a non-zero velocity component in the y-direction. In 2-D, the flow is in the x-y plane. While the default freestream direction is still along the positive x-axis, a non-zero AoA value will result in a non-zero freestream velocity in the y-direction. The SIDESLIP_ANGLE variable is unused in 2-D.

Defining reference values:

```
%
% Reference area for force coefficients (0 implies automatic calculation)
REF_AREA= 0
%
% Reference pressure (101325.0 N/m2 by default)
REF_PRESSURE= 101325.0
%
% Reference temperature (273.15 K by default)
REF_TEMPERATURE= 273.15
%
% Reference density (1.2886 Kg/m3 (air), 998.2 Kg/m3 (water))
REF_DENSITY= 1.2886
```

SU^2 accepts arbitrary reference values for flow non-dimensionalization and computing force coefficients. A reference area can be supplied by the user for the calculation of force coefficients (e.g. a trapezoidal wing area) with the REF_AREA variable. If REF_AREA is set equal to zero, as for the ONERA M6, a reference area will be automatically calculated by summing all surface normal components in the positive z-direction on the monitored markers. The values entered for REF_PRESSURE, REF_TEMPERATURE, and REF_DENSITY will be used to non-dimensionalize the flow based on the formulation explained on the non-dimensionalization page. If these three reference values are set equal to 1.0, SU^2 will perform a simulation with fully dimensional values.

Discussion of some key multigrid options:

```
% Multi-Grid Levels (0 = no multi-grid)
MGLEVEL= 2
%
% Multi-Grid Cycle (0 = V cycle, 1 = W Cycle)
MGCYCLE= 1
%
% Reduction factor of the CFL coefficient in the coarse levels
CFL_REDUCTION= 0.75
```

Multigrid is a convergence acceleration technique where the original mesh is simplified into a series of coarser meshes, and calculations are performed on all mesh levels with each solver iteration in order to provide a better residual update. It is implemented in SU^2 using an agglomeration algorithm. The user can set the number of multigrid levels using the MGLEVEL option. If this is set to zero, multigrid will be turned off, and only the original (fine) mesh will be used. An integer number of levels, up to 3, can be chosen. The ONERA M6 test case uses 2 levels of coarser meshes along with the original mesh for a total of 3 mesh levels. The type of cycle (V or W) can also be specified, and in general, while more computationally intensive, a W-cycle provides better performance. Lastly, for stability, the value of the CFL number for residual calculations on the coarser levels is reduced to 75% of the original value for this problem.

5.2.6 Running SU^2

Instructions for running this test case are given here for both serial and parallel computations. The computational mesh is rather large, so if possible, performing this case in parallel is recommended.

In Serial

The wing simulation is relatively large, but should still fit on a single core machine. To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.

2. Copy the config file (inv_ONERM6.cfg) and the mesh file (mesh_ONERAM6_inv.su2) to this directory.

3. Run the executable by entering "./SU2_CFD inv_ONERAM6.cfg" at the command line. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after reaching the specified convergence criteria.

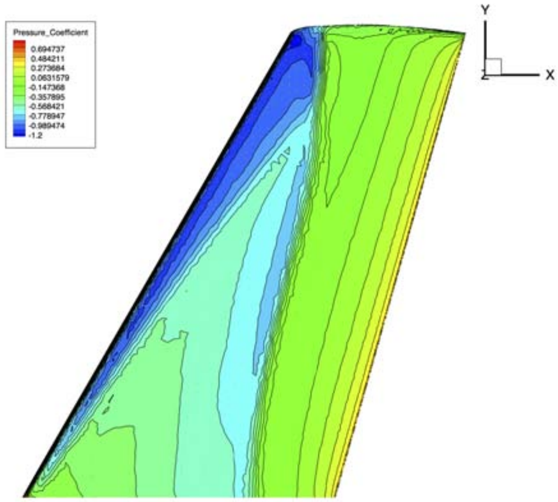


Figure 5.6: Cp contours on the upper surface of the ONERA M6.

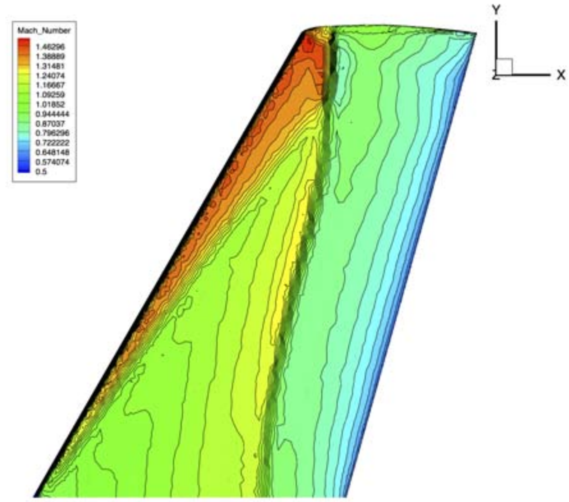


Figure 5.7: Mach number contours on the upper surface of the ONERA M6 wing. Notice the "lambda" shock pattern typically seen on the upper surface.

4. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

In Parallel

If SU^2 has been built with parallel support (note that METIS and an implementation of MPI are required for this), the recommended method for running a parallel simulation is through the use of the parallel_computation.py Python script. This automatically handles the domain decomposition with SU2_DDC, execution of SU2_CFD, and the merging of the decomposed files. Follow these steps to run the ONERA M6 case in parallel:

1. Move to the SU2/SU2Py/ directory where the parallel_computation.py script can be found.
2. Copy the config file (inv_ONERM6.cfg) and the mesh file (mesh_ONERAM6_inv.su2) to this directory.
3. Run the python script by entering "python parallel_computation.py -f inv_ONERAM6.cfg -p NP" at the command line with NP being the number of processors to be used for the simulation. Note that it is assumed that both SU2_CFD and SU2_DDC have been built and their executables exist in the SU2/SU2Py/ directory. The python script will automatically call SU2_DDC to perform the domain decomposition, followed by SU2_CFD to perform the simulation in parallel. Each mesh partition and corresponding solution file name will be appended with the partition number.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after reaching the specified convergence criteria.
5. The python script will automatically call another script for merging the decomposed solution files from each processor into a single file. These files containing the results will be written upon exiting SU^2. The flow solution can then be visualized in ParaView (.vtk) or Tecplot (.plt).

5.2.7 Results

Results are here given for the SU^2 solution of inviscid flow over the ONERA M6 wing.

5.3 Tutorial 3 - Laminar Flat Plate

5.3.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, laminar flow over a flat plate. The solution will provide a laminar boundary layer on the surface, which can be compared to the Blasius solution as a validation case for SU^2. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 2-D, Laminar Navier-Stokes equations
- Multigrid
- Roe (Second-Order) numerical scheme in space
- Euler implicit time integration
- Inlet, Outlet, Symmetry, and Navier-Stokes Wall boundary conditions

The intent of this tutorial is to introduce a common viscous test case which is used to explain how different equation sets can be chosen in SU^2. We also introduce some details on the numerics and a new type of convergence criteria which monitors the change of a specific objective, such as lift or drag, in order to assess convergence.

5.3.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/lam_FLAT_PLATE/ directory. You will need the configuration file (lam_flatplate.cfg) and the mesh file (mesh_flatplate_65x65.su2).

5.3.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow over a flat plate using SU^2. It is assumed you have already obtained and compiled the SU2_CFD code for a serial computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.3.4 Background

In his Ph.D dissertation in 1908, H. Blasius obtained what is now referred to as the Blasius equation for incompressible, laminar flow over a flat plate:

$$\begin{aligned}
&2f''' + ff'' = 0\\
&\text{At } \eta = 0 : f(\eta) = 0,\ f'(\eta) = 0\\
&\text{At } \eta \to \infty : f'(\eta) = 1
\end{aligned}$$

The third-order, ordinary differential equation can be solved numerically using a shooting method resulting in the well-known laminar boundary layer profile. Using the numerical solution, an expression for the skin friction coefficient along the flat plate can also be derived:

$$c_f \approx \frac{0.664}{\sqrt{Re_x}}$$

where Re_x is the Reynolds number along the plate. In this tutorial, we will perform a solution of nearly incompressible (low Mach number) laminar flow over a flat plate and compare our results against the analytical Blasius solutions for the profile shape and skin friction coefficient along the plate. This problem has become a classic validation case for viscous flow solvers. More detail on the Blasius solution and the similarity variables can be found in Chapter 18 of Fundamentals of Aerodynamics (Fourth Edition) by John D. Anderson, Jr and most other texts on aerodynamics.

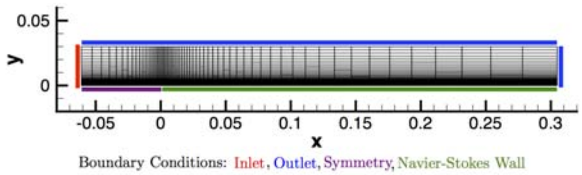


Figure 5.8: Figure of the computational mesh with boundary conditions.

5.3.5 Problem Setup

This problem will solve the for the flow over the flat plate with these conditions:

- Inlet Stagnation Temperature = 300.0 K
- Inlet Stagnation Pressure = 100000.0 N/m2
- Inlet Flow Direction, unit vector (x,y,z) = (1.0, 0.0, 0.0)
- Outlet Static Pressure = 97250.0 N/m2
- Resulting free-stream Mach number = 0.2
- Reynolds number = 1301233.166 for a plate length of 0.3048 m (1 ft)

For flow initialization, the free-stream Mach number in the configuration file will be ignored for internal flows in favor of a Mach number calculated at the inlet using isentropic relations.

Mesh Description

The computational mesh for the flat plate is structured (rectangles) with 65 nodes in both the x- and y-directions. The flat plate is along the lower boundary of the domain ($y = 0$) starting at $x = 0$ m and is of length 0.3048 m (1 ft). In the figure of the mesh, this corresponds to the Navier-Stokes boundary condition highlighted in green. The domain extends a distance upstream of the flat plate, and a symmetry boundary condition is used to simulate a free-stream approaching the plate in this region (highlighted in purple). Axial stretching of the mesh is used to aid in resolving the region near the start of the plate where the no-slip Navier-Stokes boundary condition begins at $x = 0$ m, as shown in Figure (1).

Because the flow is subsonic and disturbances caused by the presence of the plate can propagate both upstream and downstream, the characteristic-based, subsonic inlet and outlet boundary conditions are used for the flow entrance plane (red) and the outflow regions along the upper region of the domain and the exit plane at x= 0.3048 m (blue). As usual for viscous flows, the mesh spacing at the wall is important, and an appropriate level of fineness is required to capture the viscous boundary layer. In this mesh, the vertical spacing is such that approximately 30 nodes lie within the boundary layer.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Problem Definition:

```
%
% Physical governing equations (EULER, NAVIER_STOKES,
% PLASMA, TWO_PHASE_FLOW, COMBUSTION)
PHYSICAL_PROBLEM= NAVIER_STOKES
%
% If Navier-Stokes, kind of turbulent model (NONE, SA)
KIND_TURB_MODEL= NONE
```

To compute viscous flows, the Navier-Stokes governing equations are selected. In conjunction with selecting Navier-Stokes as the problem type, the type of turbulence model must also be specified. Laminar flows can be computed by entering "NONE" as the option for the type of turbulence model. For turbulent flows, SU^2 also has the Spalart-Allmaras model (SA) available. If this were an inviscid flow problem, the user would enter "EULER" for the problem type. SU^2 supports other problem types, as well, and the user is referred to the configuration page for a description of the possible options.

Numerics:

```
%
% Numerical method for spatial gradients (GREEN_GAUSS,
% WEIGHTED_LEAST_SQUARES)
NUM_METHOD_GRAD= WEIGHTED_LEAST_SQUARES
%
% Courant-Friedrichs-Lewy condition of the finest grid
CFL_NUMBER= 4.0
%
% CFL ramp (factor, number of iterations, CFL limit)
CFL_RAMP= ( 4.0, 50, 1025.0 )
%
% Convective numerical method (JST, LAX-FRIEDRICH,
% ROE-1ST_ORDER, ROE-2ND_ORDER)
CONV_NUM_METHOD_FLOW= ROE-2ND_ORDER
%
% Slope limiter (NONE, VENKATAKRISHNAN)
SLOPE_LIMITER_FLOW= NONE
%
% Viscous numerical method (AVG_GRAD, AVG_GRAD_CORRECTED)
VISC_NUM_METHOD_FLOW= AVG_GRAD
```

For laminar flow, these options for the numerics showed good performance. The gradients are calculated via weighted least squares. Implicit time integration allowed for not only a starting CFL number of 4, but also very aggressive ramping of the CFL number during the computation. The format for the CFL ramping is (multiplicative factor, number of iterations between applying factor, maximum CFL value). For our options, the starting CFL of 4 will be multiplied by a factor of 4 every 50 iterations until reaching the upper limit of 1025.0. The 2nd-order Roe upwinding method without a limiter is used for computing fluxes, and the viscous terms are computed with the average of gradients method.

Convergence:

```
%
% Convergence criteria (CAUCHY, RESIDUAL)
CONV_CRITERIA= CAUCHY
%
% Number of elements to apply the criteria
CAUCHY_ELEMS= 100
%
% Epsilon to control the series convergence
CAUCHY_EPS= 1E-6
%
% Function to apply the criteria (LIFT, DRAG, NEARFIELD_PRESS,
% SENS_GEOMETRY, SENS_MACH, DELTA_LIFT, DELTA_DRAG)
CAUCHY_FUNC_FLOW= DRAG
```

Rather than achieving a certain order of magnitude in the density residual to judge convergence, what we call the Cauchy convergence criteria is chosen for this problem. This type of criteria measures the change in a specific quantity of interest over a specified number of previous iterations. With the options selected above, the flat plate solution will terminate when the change in the drag coefficient (CAUCHY_FUNC_FLOW) for the plate over the previous 100 iterations (CAUCHY_ELEMS) becomes less than 1E-6 (CAUCHY_EPS). A convergence criteria of this nature can be very useful for design problems where the solver is embedded in

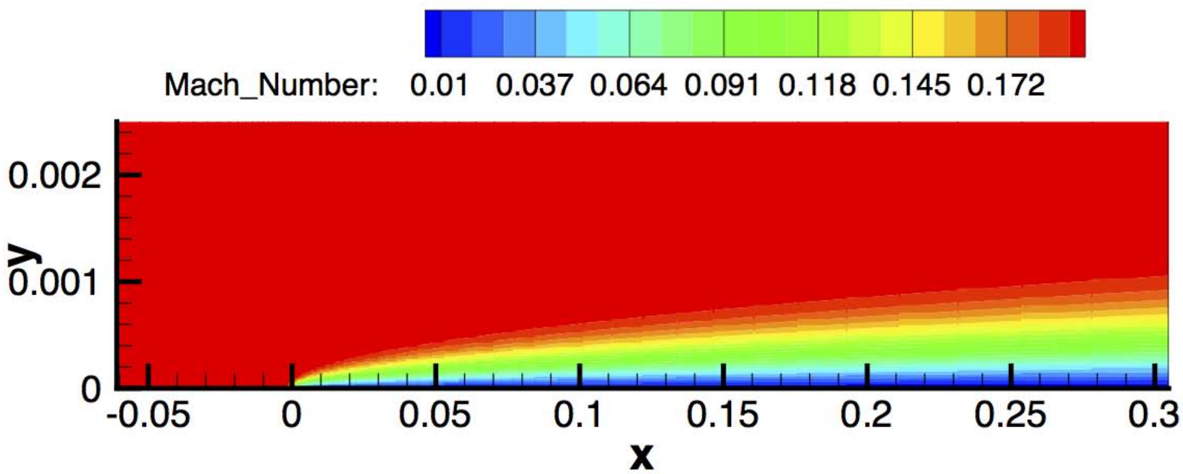


Figure 5.9: Mach contours for the laminar flat plate.

a larger design loop and reliable convergence behavior is essential. In our experience, this criteria is more robust for some problems where the residuals may continue fluctuating or plateau despite a resolved solution.

5.3.6 Running SU^2

The flat plate simulation for the 65x65 node mesh is small and will execute relatively quickly on a single workstation or laptop in serial. To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
2. Copy the config file (lam_flatplate.cfg) and the mesh file (mesh_flatplate_65x65.su2) to this directory.
3. Run the executable by entering "./SU2_CFD lam_flatplate.cfg" at the command line.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after reaching the specified convergence criteria.
5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.3.7 Results

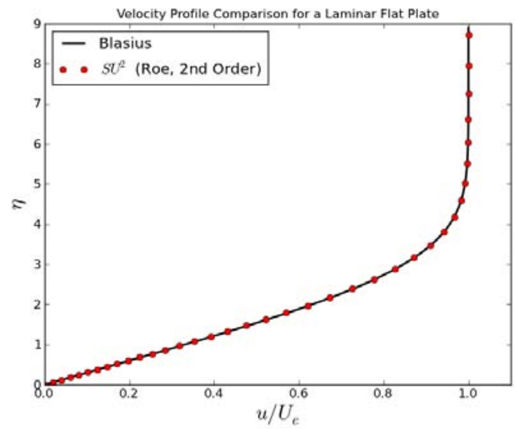
Results are here given for the SU^2 solution of laminar flow over the flat plate. The results show excellent agreement with the analytical Blasius solution.

5.4 Tutorial 4 - Laminar Cylinder

5.4.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, laminar flow around a 2-D geometry. The specific geometry chosen for the tutorial is a cylinder. Consequently, the following capabilities of SU^2 will be showcased:

- Steady, 2-D Laminar Navier-Stokes equations
- Multigrid
- Roe (Second-Order) numerical scheme in space



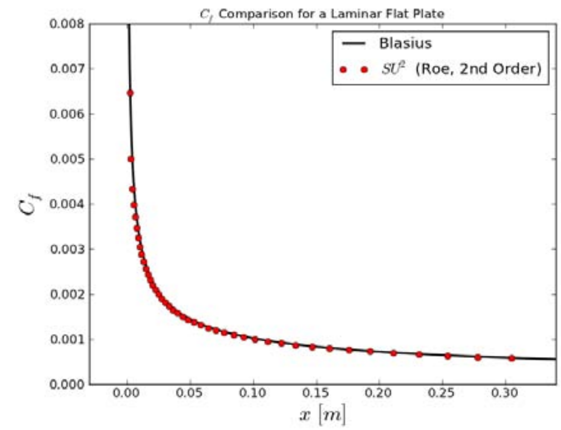


Figure 5.10: Velocity data was extracted from the exit plane of the mesh (x = 0.3048 m) near the wall, and the boundary layer velocity profile was plotted compared to and using the similarity variables from the Blasius solution.

Figure 5.11: A plot of the skin friction coefficient along the plate created using the values written in the surface_flow.csv file and compared to Blasius.

- Euler implicit time integration
- Navier-Stokes Wall and Farfield boundary conditions

In this tutorial, we discuss some numerical method options, including how to activate a slope limiter for upwind methods.

5.4.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/lam_CYLINDER/ directory. You will need the configuration file (lam_cylinder.cfg) and the mesh file (mesh_cylinder_lam.su2).

5.4.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow around a cylinder using SU^2. It is assumed you have already obtained and compiled the SU2_CFD code for a serial computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.4.4 Background

The flow around a (geometrically) two-dimensional circular cylinder is case that has been used both as a validation case and as a legitimate research case. At very low Reynolds numbers of less than 46, the flow is steady and symmetrical. As the Reynolds number is increased, asymmetries and time-dependence develop, eventually resulting in the Von Karmann vortex street, and then on to turbulence.

5.4.5 Problem Setup

This problem will solve the for the external flow over the cylinder with these conditions:

- Freestream temperature = 273.15 K
- Freestream Mach number = 0.3
- Angle of attack (AoA) = 0.0 degrees
- Reynolds number = 40 for a cylinder radius of 0.3048 m (1 ft)

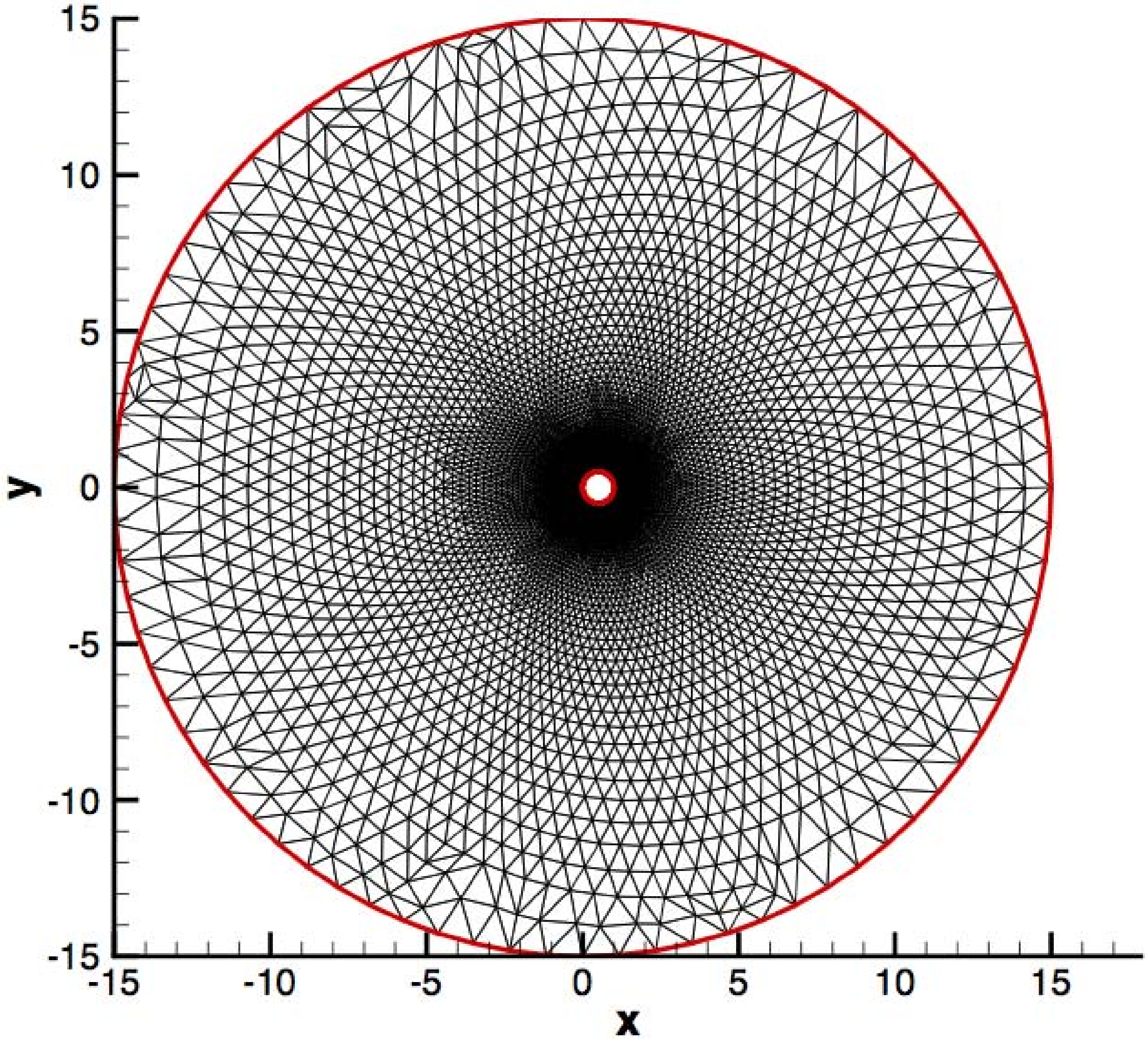


Figure 5.12: The computational mesh for the 2-D cylinder test case. The outer boundary in red is the far-field, and the small circle in the center is the cylinder which uses the Navier-Stokes Wall boundary condition.

Mesh Description

The problem geometry is two-dimensional.The mesh has 26,192 triangular elements. It is fine near the surface of the cylinder to resolve the boundary layer. The exterior boundary is approximately 15 diameters away from the cylinder surface to avoid interaction between the boundary conditions. Far-field boundary conditions are used at the outer boundary. No-slip boundary conditions are placed on the surface of the cylinder.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Some options concerning numerics:

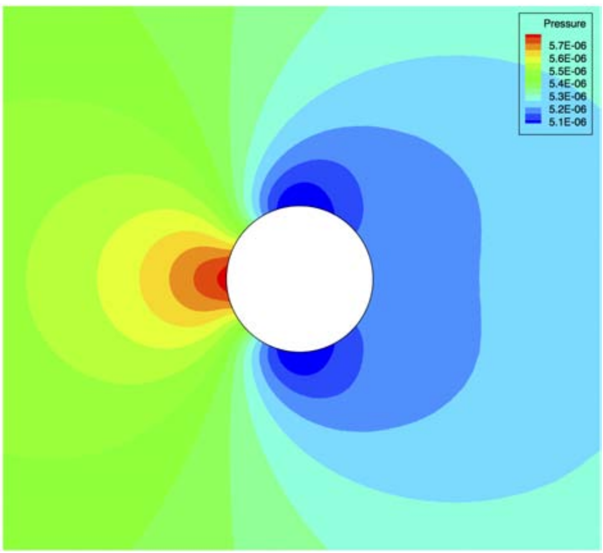


Figure 5.13: Pressure contours around the cylinder.

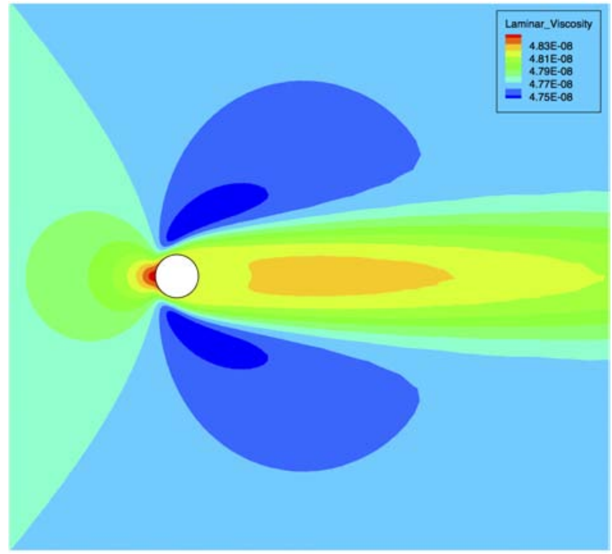


Figure 5.14: Laminar viscosity contours for this steady, low Reynolds number flow.

```
%
% Convective numerical method (JST, LAX-FRIEDRICH,
% ROE-1ST_ORDER, ROE-2ND_ORDER)
CONV_NUM_METHOD_FLOW= ROE-2ND_ORDER
%
% Slope limiter (NONE, VENKATAKRISHNAN)
SLOPE_LIMITER_FLOW= VENKATAKRISHNAN
%
% Viscous numerical method (AVG_GRAD, AVG_GRAD_CORRECTED)
VISC_NUM_METHOD_FLOW= AVG_GRAD_CORRECTED
```

For laminar flow around the cylinder, the 2nd-order Roe upwinding method showed good performance when the Venkatakrishnan limiter was used. Without the limiter, the computation is much less stable and may not converge. The viscous terms are computed with the corrected average of gradients method.

5.4.6 Running SU^2

The cylinder simulation for the 13,336 node mesh is small and will execute relatively quickly on a single workstation or laptop in serial. To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
2. Copy the config file (lam_cylinder.cfg) and the mesh file (mesh_cylinder_lam.su2) to this directory.
3. Run the executable by entering "./SU2_CFD lam_cylinder.cfg" at the command line. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after meeting the specified convergence criteria.
4. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.4.7 Results

The following results show the flow around the cylinder as calculated by SU^2.

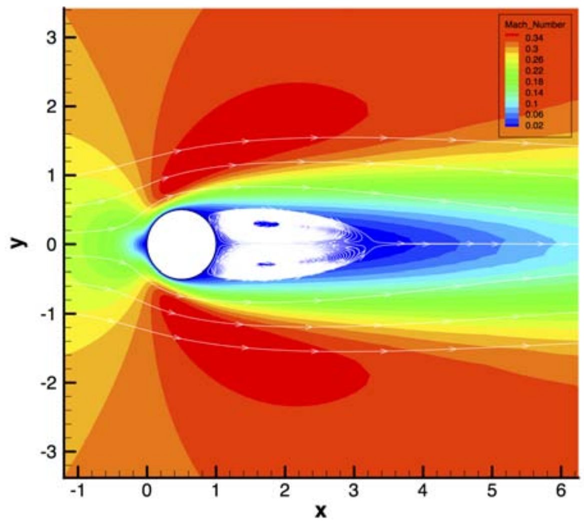


Figure 5.15: Mach number contours around the cylinder with streamlines. Note the large laminar separation region behind the cylinder at Re = 40.

5.5 Tutorial 5 - Turbulent Flat Plate

5.5.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, turbulent flow over a flat plate. Consequently, the following capabilities of SU^2 will be showcased and validated against experimental data in this tutorial:

- Steady, 2-D RANS Navier-Stokes equations
- Spalart-Allmaras turbulence model
- Multigrid
- Roe 2nd order numerical scheme in space
- Euler implicit time integration
- Inlet, Outlet, and Navier-Stokes Wall boundary conditions

In this tutorial, we perform our first RANS simulation with the Spalart-Allmaras (SA) turbulence model.

5.5.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/turb_FLAT_PLATE/ directory. You will need the configuration file (turb_flatplate.cfg) and either of the two available mesh files (mesh_flatplate_turb_137x97.su2 or mesh_flatplate_turb_545x385.su2).

Additionally, skin friction and velocity profiles corresponding to this testcase (obtained from the Langley Research Center Turbulence Modeling Resource website shown below) are used for later comparison with SU^2 results. These files can be found on the following website: http://turbmodels.larc.nasa.gov/flatplate.html

5.5.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow around a cylinder using SU^2. It is assumed you have already obtained and compiled the SU2_CFD code for a serial computation or both the SU2_CFD and SU2_DDC codes for a parallel computation. If you have yet to complete these requirements, please see the Download and Installation pages.

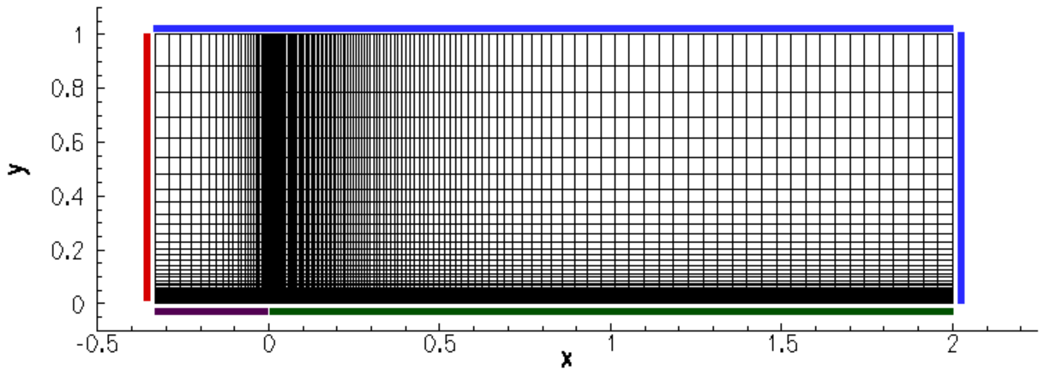


Figure 5.16: Mesh with boundary conditions (inlet, outlet, symmetry, wall).

5.5.4 Background

Turbulent flow over a zero pressure gradient flat plate is a common testcase for the verification and validation of turbulence models in CFD solvers. The flow is everywhere turbulent and a boundary layer develops over the surface of the flat plate. The lack of separation bubbles or other more complex flow phenomena allows turbulence models to predict the flow with a high level of accuracy. Due to the low Mach number of 0.2 compressibility effects are nonexistent.

For verification purposes we will be comparing SU^2 results against those from the NASA codes FUN3D and CFL3D. For validation purposes we'll compare profiles of u+ vs. y+ against theoretical profiles of the viscous sublayer and log law region.

5.5.5 Problem Setup

The length of the flat plate is 2m, and is represented by an adiabatic no-slip wall boundary condition. Also part of the domain is a symmetry plane located before the leading edge of the flat plate. Inlet and outlet boundary conditions are used on the left and right boundaries of the domain, and an outlet boundary condition is used over the top region of the domain, which is located 1m away from the flat plate.

The Reynolds number based on a length of 1m is 5 million, and the Mach number is 0.2.

Mesh Description

The mesh used for this tutorial, which consists of 13,056 rectangular elements, is shown below.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

For the first time in the tutorials, we will use a turbulence model:

```
%
% Physical governing equations (EULER, NAVIER_STOKES,
% PLASMA, TWO_PHASE_FLOW, COMBUSTION)
PHYSICAL_PROBLEM= NAVIER_STOKES
%
% If Navier-Stokes, kind of turbulent model (NONE, SA)
KIND_TURB_MODEL= SA
```

The governing equations are Navier-Stokes, but by entering "SA" as the option for "KIND_TURB_MODEL," we activate the RANS governing equations with the Spalart-Allmaras (SA) turbulence model. The SA model is composed of one-equation for a turbulence field variable that is directly related to the turbulent eddy viscosity. It is a popular choice for external aerodynamic flows, such as those around airfoils and wings. In

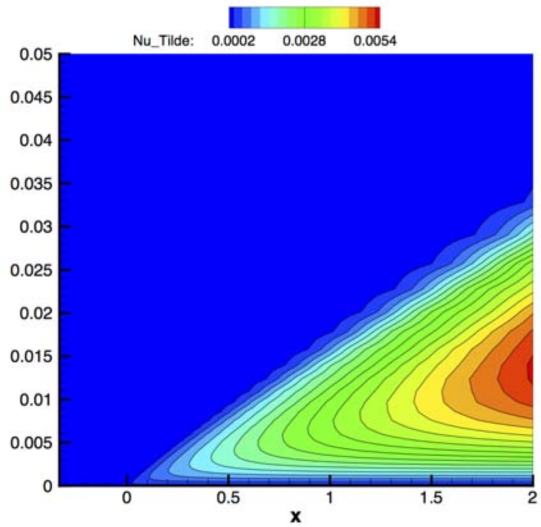


Figure 5.17: Contour of turbulence variable (nu-hat).

previous tutorials, "NONE" has been chosen, resulting in the use of the laminar Navier-Stokes governing equations.

5.5.6 Running SU^2

To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.

2. Copy the config file (turb_flatplate.cfg) and the mesh file (mesh_flatplate_turb_137x97.su2) to this directory.

3. Run the executable by entering "./SU2_CFD turb_flatplate.cfg" at the command line.

4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish upon reaching the specified convergence criteria.

5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.5.7 Results

The figures below show results obtained from SU^2 and compared to several results from NASA codes. Note that the SU^2 results for the skin friction correspond to the coarser mesh (mesh_flatplate_turb_137x97.su2) while the NASA results are based on the finer mesh (mesh_flatplate_turb_545x385.su2). SU^2 still matches very closely.

5.6 Tutorial 6 - Turbulent RAE 2822

5.6.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, viscous flow around a 2-D geometry. The specific geometry chosen for the tutorial is the RAE 2822 transonic airfoil, and it will also double as a validation case for SU^2. Consequently, the following capabilities of SU^2 will be showcased and validated against experimental data in this tutorial:

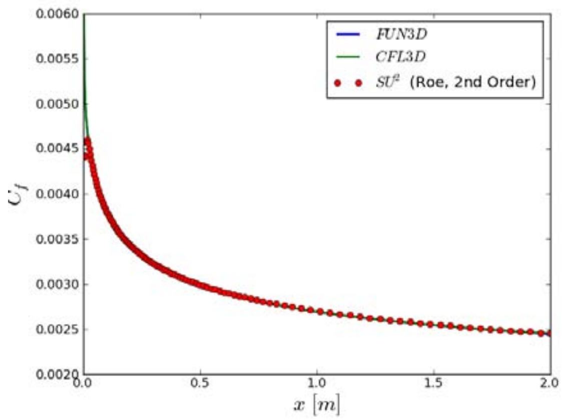


Figure 5.18: Profile for the skin friction coefficient.

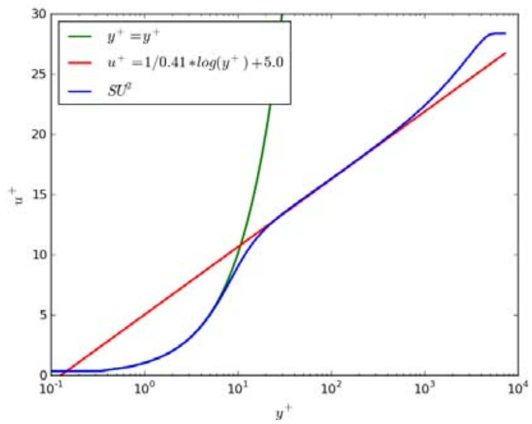


Figure 5.19: Velocity profile comparison against law of the wall.

- Steady, 2-D RANS equations
- Spalart-Allmaras turbulence model
- Multigrid
- JST numerical scheme in space
- Backward Euler (implicit) time integration
- Navier-Stokes Wall and Farfield boundary conditions

In this tutorial, we will discuss the numerical method options for solving the SA turbulence equation.

5.6.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/turb_RAE2822/ directory. You will need the configuration file (turb_RAE2822.cfg) and the mesh file (mesh_RAE2822_turb.su2).

Additionally, Cp distributions obtained from experiments are available at the NASA link shown below will be used for later comparison with SU^2 results.

5.6.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow around the RAE 2822 airfoil using SU^2. It is assumed you have already obtained and compiled the SU2_CFD code for a serial computation or both the SU2_CFD and SU2_DDC codes for a parallel computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.6.4 Background

The RAE 2822 airfoil is a supercritical airfoil commonly used for the validation of turbulence models. For this test case the flow is fully two dimensional, turbulent, and transonic. Additionally, conditions are such that no separation occurs downstream of the shock position.

The test case is based on the RAE 2822 Transonic Airfoil Study # 4 of NASA's NPARC Alliance Verification and Validation Archive, which can be found here: NASA Validation.

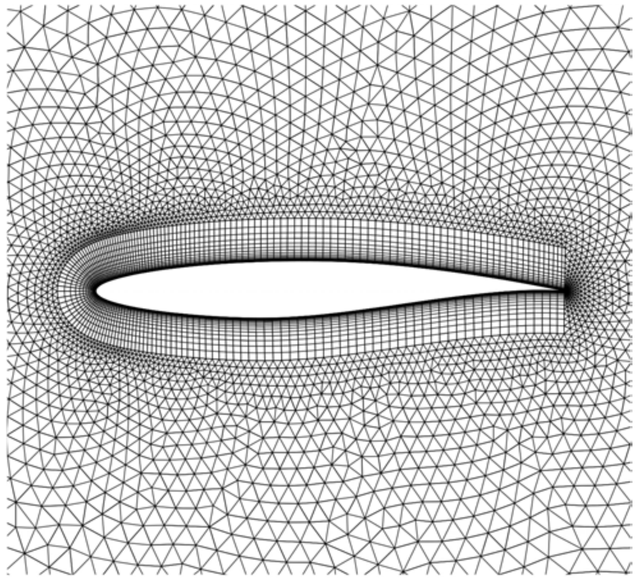


Figure 5.20: Close-up view of the hybrid mesh near the RAE 2822 surface.

5.6.5 Problem Setup

The Reynolds number is 6.5 million based on a unit chord length, the Mach number is 0.729, and the airfoil is inclined at an angle of attack of 2.31 degrees. Two boundary conditions are required: a viscous wall boundary condition used over the surface of the airfoil, and a freestream boundary condition used at the outer edge of the domain where the flow is everywhere subsonic. The flow is initialized using free stream values.

Mesh Description

The mesh used for this tutorial (contained in the file mesh_RAE2822_turb.su2) is an unstructured O-grid that wraps around the RAE2822 airfoil. It has 22,842 elements, out of which 192 constitute the airfoil boundary and 40 constitute the farfield boundary. The mesh is a hybrid one, with quadrilaterals in the region adjacent to the airfoil surface and triangles in the remaining portion of the computational domain. The farfield boundary is located approximately one hundred chord lengths away from the airfoil. The figure below shows the mesh used for the tutorial.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

We have already discussed some options for the numerics of the mean flow, so we will now consider the options for the turbulent numerical method:

```
%
% Convective numerical method (SCALAR_UPWIND-1ST_ORDER,
% SCALAR_UPWIND-2ND_ORDER)
CONV_NUM_METHOD_TURB= SCALAR_UPWIND-1ST_ORDER
%
% Slope limiter (NONE, VENKATAKRISHNAN)
SLOPE_LIMITER_TURB= NONE
%
% Viscous numerical method (AVG_GRAD, AVG_GRAD_CORRECTED)
VISC_NUM_METHOD_TURB= AVG_GRAD_CORRECTED
%
% Source term numerical method (PIECEWISE_CONSTANT)
SOUR_NUM_METHOD_TURB= PIECEWISE_CONSTANT
%
% Time discretization (EULER_IMPLICIT)
TIME_DISCRE_TURB= EULER_IMPLICIT
```

These options control how the one-equation Spalart-Allmaras turbulence model is solved numerically. Two upwind methods are available for solving the convective terms of the scalar equation which offer first- and second-order accuracy in space. For the RAE 2822, the first-order method is set for the CONV_NUM_METHOD_TURB option. The VENKATAKRISHNAN slope limiter can also be applied to the upwind methods. Viscous terms are calculated using the corrected average of gradients method (AVG_GRAD_CORRECTED). Source terms are approximated using piecewise constant reconstruction within each of the finite volume cells. The only time integration method available for the turbulence equation is Euler implicit, which is chosen for the TIME_DISCRE_TURB option.

5.6.6 Running SU^2

This test case can be executed relatively quickly in serial. To run it, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
2. Copy the config file (turb_RAE2822.cfg) and the mesh file (mesh_RAE2822_turb.su2) to this directory.
3. Run the executable by entering "./SU2_CFD turb_RAE2822.cfg" at the command line.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria.
5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.6.7 Results

The figures below show the results obtained from SU^2 for turbulent flow around the RAE 2822.

5.7 Tutorial 7 - Turbulent ONERA M6

5.7.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of external, viscous flow around a 3-D geometry using a turbulence model. The specific geometry chosen for the tutorial is the classic ONERA M6 wing. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 3-D RANS equations
- Spalart-Allmaras turbulence model

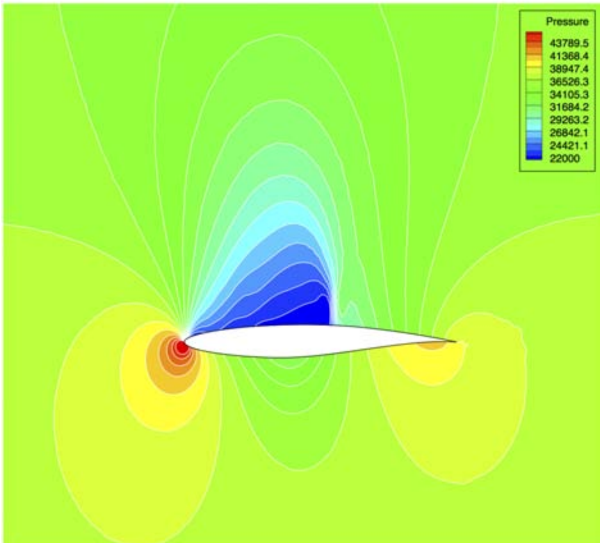


Figure 5.21: Pressure contours around the RAE 2822.

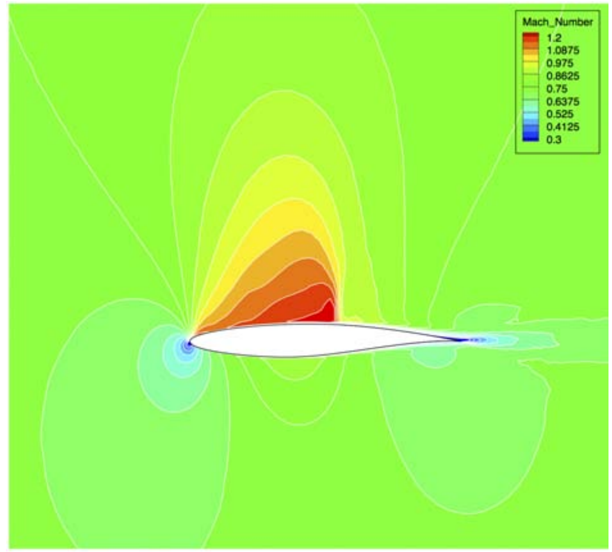


Figure 5.22: Mach number contours.

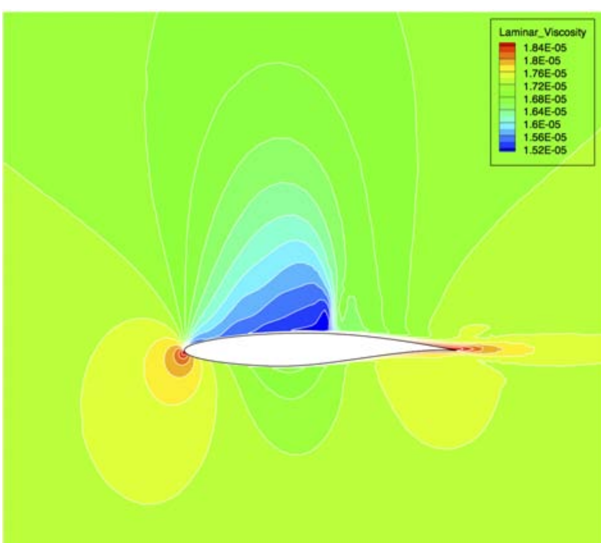


Figure 5.23: Laminar viscosity contours.

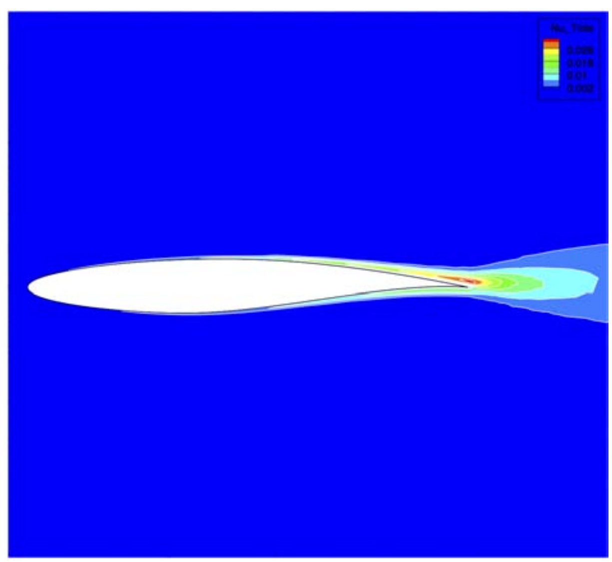


Figure 5.24: SA turbulence variable contours.

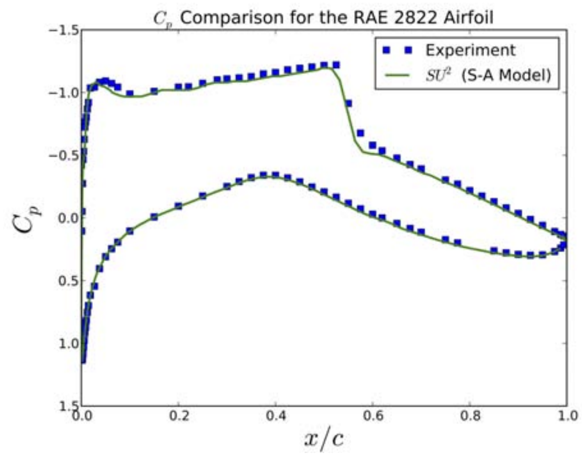


Figure 5.25: Pressure coefficient comparison with experimental data.

- Multigrid
- JST numerical scheme in space
- Euler implicit time integration
- Navier-Stokes Wall, Symmetry, and Farfield boundary conditions
- Code parallelism (optional)

This tutorial also provides an explanation for properly setting up viscous, 3-D flow conditions in SU^2.

5.7.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/turb_ONERAM6/ directory. You will need the configuration file (turb_ONERAM6.cfg) and the mesh file (mesh_ONERAM6_turb.su2).

5.7.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow around the ONERA M6 using SU^2. The tutorial will also address procedures for both serial and parallel computations. To this end, it is assumed you have already obtained and compiled the SU2_CFD code for a serial computation or both the SU2_CFD and SU2_DDC codes for a parallel computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.7.4 Background

This test case is for the ONERA M6 wing in viscous flow. The ONERA M6 wing was designed in 1972 by the ONERA Aerodynamics Department as an experimental geometry for studying three-dimensional, high Reynolds number flows with some complex flow phenomena (transonic shocks, shock-boundary layer interaction, separated flow). It has become a classic validation case for CFD codes due to the simple geometry, complicated flow physics, and availability of experimental data. This particular study will be performed at a transonic Mach number with the 3-D RANS equations in SU^2.

5.7.5 Problem Setup

This problem will solve the for the flow past the wing with these conditions:

- Freestream Temperature = 273.15 K
- Freestream Mach number = 0.8395

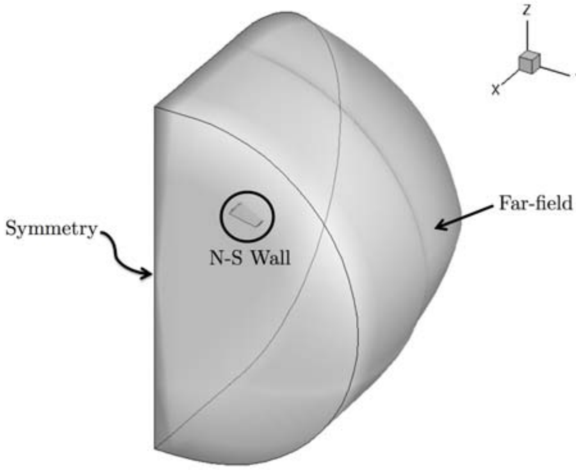


Figure 5.26: Far-field view of the computational mesh.

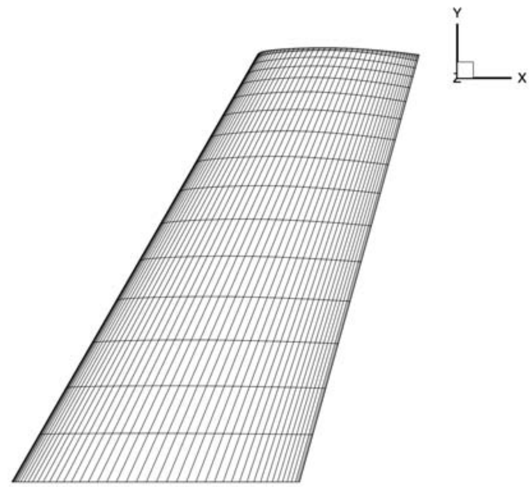


Figure 5.27: Close-up view of the structured surface mesh on the upper wing surface.

- Angle of attack (AoA) = 3.06 deg
- Reynolds number = 11720000.0
- Reynolds length = 1.0 m

These transonic flow conditions will cause the typical ”lambda” shock along the upper surface of the lifting wing.

Mesh Description

The computational domain is a large C-type mesh with the wing half-span on one boundary in the x-z plane. The mesh consists of 43,008 interior elements and 46,417 nodes. Three boundary conditions are employed: the Navier-Stokes wall condition on the wing surface, the far-field characteristic-based condition on the far-field markers, and a symmetry boundary condition for the marker where the wing half-span is attached. The symmetry condition acts to mirror the flow about the x-z plane, reducing the complexity of the mesh and the computational cost. Images of the entire domain and the structured, rectangular elements on the wing surface are shown below.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Setting the viscous, 3-D flow conditions:

```
%
% Mach number in the farfield
MACH_NUMBER= 0.8395
%
% Angle of attack (degrees)
AoA= 3.06
%
% Side-slip angle (degrees)
SIDESLIP_ANGLE= 0.0
%
% Ratio of specific heats
GAMMA_VALUE= 1.4
%
% Specific gas constant ( J/(kg*K) )
GAS_CONSTANT= 287.87
%
% Total temperature (Kelvin)
FREESTREAM_TEMPERATURE= 273.15
%
% Reynolds number (0.0 implies no definition)
REYNOLDS_NUMBER= 11720000.0
%
% Reynolds length (dimensional)
REYNOLDS_LENGTH= 1.0
```

The options above set the conditions for a 3-D, viscous flow. The MACH_NUMBER, AoA, and SIDESLIP_ANGLE options remain the same as they appeared for the inviscid ONERA M6 tutorial, which includes a description of the freestream flow direction. For the RANS equations, SU^2 assumes a calorically perfect working gas. The ratio of specific heats and specific gas constant can be explicitly chosen with the GAMMA_VALUE and GAS_CONSTANT options, respectively. Most importantly for a viscous simulation, the numerical experiment must match the physical reality. This flow similarity is achieved by matching the REYNOLDS_NUMBER and REYNOLDS_LENGTH to the original system (assuming the Mach number and the geometry already match). Upon starting a viscous simulation in SU2, the following steps are performed to set the flow conditions:

1. Store the gas constants and freestream temperature, and calculate the speed of sound.
2. Calculate and store the freestream velocity vector from the Mach number, AoA/sideslip angle, and speed of sound from step 1.
3. Compute the freestream viscosity from Sutherland's law and the supplied freestream temperature.
4. Use the definition of the Reynolds number to find the freestream density from the supplied Reynolds information, freestream velocity, and freestream viscosity from step 3.
5. Calculate the freestream pressure using the perfect gas law with the freestream temperature, specific gas constant, and freestream density from step 4.
6. Perform any required non-dimensionalization if reference values are not equal to 1, and initialize the entire flow field with these quantities.

Notice that the freestream pressure supplied in the configuration file will be ignored for viscous computations. Lastly, it is important to note that this method for setting similar flow conditions requires that all inputs are in SI units, including the mesh geometry. If your mesh is not in meters, or needs to be scaled in some way to match the flow conditions, the CONVERT_TO_METER option can be used:

```
%
% Conversion factor for converting the grid to meters
CONVERT_TO_METER= 1.0
```

For the ONERA M6, the mesh is already in meters, so no conversion is necessary. If your mesh requires conversion, enter a non-zero value for this option, and every node in the mesh will be multiplied by this factor upon reading and storing the mesh. For example, to halve the size of the mesh, enter CONVERT_TO_METER = 0.5. At the end of a simulation, the mesh will not be converted back to the original size. Therefore, all solution files will contain the solution and geometry in the converted state.

5.7.6 Running SU^2

Instructions for running this test case are given here for both serial and parallel computations.

In Serial

The wing mesh should easily fit on a single core machine. To run this test case, follow these steps at a terminal command line:

- Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
- Copy the config file (turb_ONERM6.cfg) and the mesh file (mesh_ONERAM6_turb.su2) to this directory.
- Run the executable by entering "./SU2_CFD turb_ONERAM6.cfg" at the command line.
- SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after reaching the specified convergence criteria.
- Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

In Parallel

If SU^2 has been built with parallel support (note that METIS and an implementation of MPI are required for this), the recommended method for running a parallel simulation is through the use of the parallel_computation.py Python script. This automatically handles the domain decomposition with SU2_DDC, execution of SU2_CFD, and the merging of the decomposed files. Follow these steps to run the ONERA M6 case in parallel:

1. Move to the SU2/SU2Py/ directory where the parallel_computation.py script can be found.
2. Copy the config file (turb_ONERM6.cfg) and the mesh file (mesh_ONERAM6_turb.su2) to this directory.
3. Run the python script by entering "python parallel_computation.py -f turb_ONERAM6.cfg -p NP" at the command line with NP being the number of processors to be used for the simulation. Note that it is assumed that both SU2_CFD and SU2_DDC have been built and their executables exist in the SU2/SU2Py/ directory. The python script will automatically call SU2_DDC to perform the domain decomposition, followed by SU2_CFD to perform the simulation in parallel. Each mesh partition and corresponding solution file name will be appended with the partition number.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will terminate after reaching the specified convergence criteria.
5. The python script will automatically call another script for merging the decomposed solution files from each processor into a single file. These files containing the results will be written upon exiting SU^2. The flow solution can then be visualized in ParaView (.vtk) or Tecplot (.plt).

5.7.7 Results

Results are here given for the SU^2 solution of turbulent flow over the ONERA M6 wing.

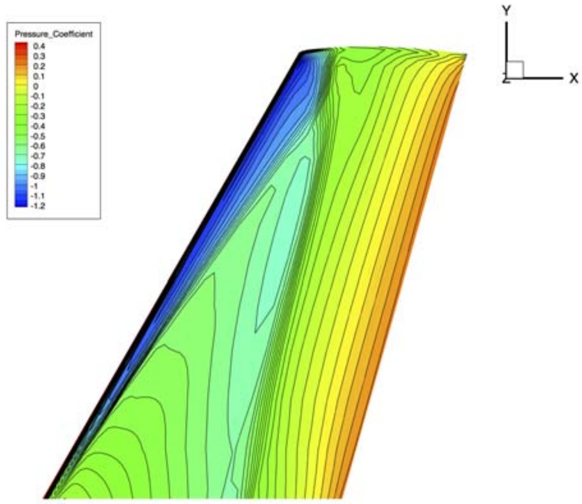


Figure 5.28: Pressure contours on the upper surface of the ONERA M6.

5.8 Tutorial 8 - Optimal Shape Design of a Rotating Airfoil

5.8.1 Goals

Upon completing this tutorial, the user will be familiar with performing an optimal shape design of a 2-D geometry. The initial geometry chosen for the tutorial is a NACA 0012 airfoil that is rotating at transonic speed in inviscid fluid. This tutorial is mean to be an introduction for using the components of SU^2 for shape design. Consequently, the following SU^2 tools will be showcased in this tutorial:

- SU2_CFD - performs the direct and the adjoint flow simulations
- SU2_GPC - projects the adjoint surface sensitivities into the design space to obtain the gradient
- SU2_MDC - deforms the geometry and mesh with changes in the design variables during the shape optimization process
- shape_optimization.py - automates the entire shape design process by executing the SU^2 tools and optimizer

5.8.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/rot_NACA0012/ directory. You will need the configuration file (rot_NACA0012.cfg) and the mesh file (mesh_NACA0012_rot.su2).

5.8.3 Tutorial

The following tutorial will walk you through the steps required when performing shape design for the rotating airfoil using SU^2. It is assumed that you have already obtained and compiled SU2_CFD, SU2_GPC, and SU2_MDC. The design loop is driven by the shape_optimization.py script, and thus Python along with the NumPy and SciPy Python modules are required for this tutorial. If you have yet to complete these requirements, please see the Download and Installation pages.

5.8.4 Background

This example uses a 2-D airfoil geometry (initially the NACA 0012) which is rotating counter-clockwise in still air.

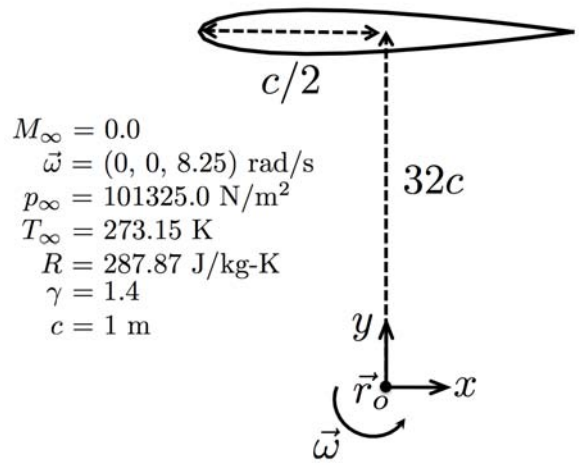


Figure 5.29: Details for the rotating airfoil numerical experiment.

5.8.5 Problem Setup

This numerical experiment for the rotating airfoil was set up such that transonic shocks would appear on the upper and lower surfaces causing drag. The goal of the design process is to minimize the coefficient of drag (Cd) by changing the shape of the airfoil without any constraints. In other words, we would like to eliminate the shocks along the airfoil surface. The details of the rotating airfoil experiment are given in Figure (1).

Mesh Description

The mesh from the Quick Start Tutorial is used again here as the initial geometry. It consists of a far-field boundary and an Euler wall along the airfoil shape. The specific airfoil is the NACA 0012, and more information on this airfoil can be found in the Quick Start Tutorial. The mesh can be seen in Figure (2).

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Rotating frame specification:

```
ROTATING_FRAME= YES
ROTATIONAL_ORIGIN= ( 0.5, -32.0, 0.0 )
ROTATION_RATE= ( 0.0, 0.0, 8.25 )
```

In SU^2, the Euler equations have been transformed into a rotating reference frame which offers an efficient, steady solution method for flows around rotating bodies in axisymmetric flow. A simulation can be executed in a rotating frame by setting the ROTATIONAL_FRAME flag to "YES." Two additional pieces of data must be supplied: the location of the rotation center (x,y,z) in the coordinate system of the computational mesh, and the angular velocity (rotation rate around the x-axis, rotation rate around the y-axis, rotation rate around the z-axis). For the rotating airfoil problem, the airfoil has a chord of 1 meter and the origin of the coordinate system (0,0,0) is at the leading edge of the airfoil. We set the rotation center to be 32 chord lengths below the center of the airfoil. The angular velocity is in the z-direction (out of the page) and has the units of radians per second.

Optimal shape design specification:

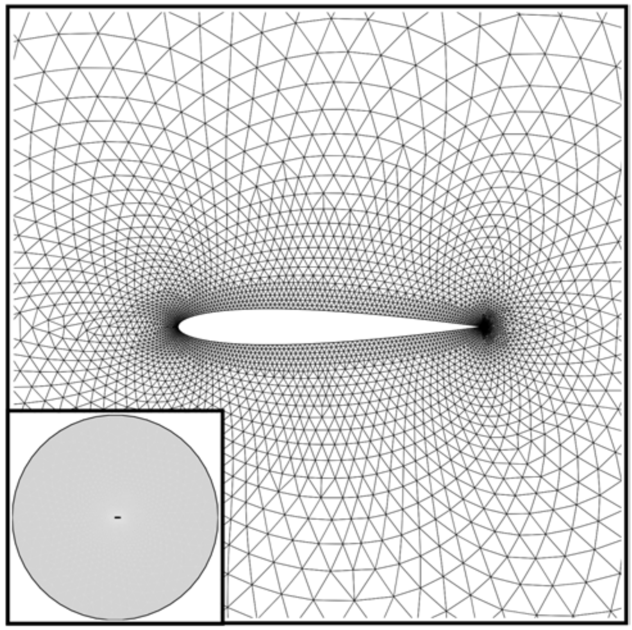


Figure 5.30: Far-field and zoom view of the initial computational mesh.

```
OBJFUNC= DRAG
CONSTRAINT= NONE
DEFINITION_DV= ( 1, 1.0 — airfoil — 0, 0.05 );
( 1, 1.0 — airfoil — 0, 0.10 ); ( 1, 1.0 — airfoil — 0, 0.15 );
( 1, 1.0 — airfoil — 0, 0.20 ); ( 1, 1.0 — airfoil — 0, 0.25 );
( 1, 1.0 — airfoil — 0, 0.30 ); ( 1, 1.0 — airfoil — 0, 0.35 );
( 1, 1.0 — airfoil — 0, 0.40 ); ( 1, 1.0 — airfoil — 0, 0.45 );
( 1, 1.0 — airfoil — 0, 0.50 ); ( 1, 1.0 — airfoil — 0, 0.55 );
( 1, 1.0 — airfoil — 0, 0.60 ); ( 1, 1.0 — airfoil — 0, 0.65 );
( 1, 1.0 — airfoil — 0, 0.70 ); ( 1, 1.0 — airfoil — 0, 0.75 );
( 1, 1.0 — airfoil — 0, 0.80 ); ( 1, 1.0 — airfoil — 0, 0.85 );
( 1, 1.0 — airfoil — 0, 0.90 ); ( 1, 1.0 — airfoil — 0, 0.95 );
( 1, 1.0 — airfoil — 1, 0.05 ); ( 1, 1.0 — airfoil — 1, 0.10 );
( 1, 1.0 — airfoil — 1, 0.15 ); ( 1, 1.0 — airfoil — 1, 0.20 );
( 1, 1.0 — airfoil — 1, 0.25 ); ( 1, 1.0 — airfoil — 1, 0.30 );
( 1, 1.0 — airfoil — 1, 0.35 ); ( 1, 1.0 — airfoil — 1, 0.40 );
( 1, 1.0 — airfoil — 1, 0.45 ); ( 1, 1.0 — airfoil — 1, 0.50 );
( 1, 1.0 — airfoil — 1, 0.55 ); ( 1, 1.0 — airfoil — 1, 0.60 );
( 1, 1.0 — airfoil — 1, 0.65 ); ( 1, 1.0 — airfoil — 1, 0.70 );
( 1, 1.0 — airfoil — 1, 0.75 ); ( 1, 1.0 — airfoil — 1, 0.80 );
( 1, 1.0 — airfoil — 1, 0.85 ); ( 1, 1.0 — airfoil — 1, 0.90 );
( 1, 1.0 — airfoil — 1, 0.95 )
```

Here we define the objective function for the optimization as drag without any constraints. It is possible, for instance, to add a lift constraint. The DEFINITION_DV is the list of design variables. For the rotating airfoil problem, we want to minimize the drag by changing the surface profile shape. To do so, we define a set of Hicks-Henne bump functions. Each design variable is separated by a semicolon. The first value in the parentheses is the variable type which is 1 for a Hicks-Henne bump function. The second value is the scale of the variable. The name between the vertical bars is the marker tag where the variable deformations will be applied. Only the airfoil surface will be deformed in this problem. The final two values in the parentheses are whether the bump function is applied to the upper (1) or lower (0) side and the x-location of the bump, respectively. Note that other types of design variables have their own specific input format.

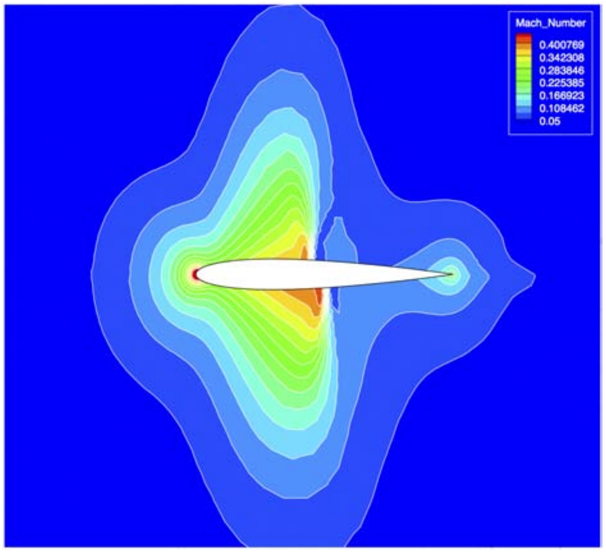


Figure 5.31: Mach number contours for the airfoil rotating in still air.

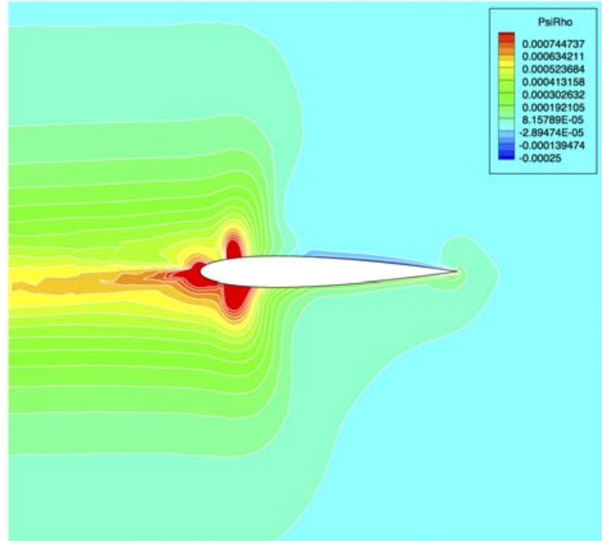


Figure 5.32: Adjoint density contours.

5.8.6 Running SU^2

A continuous adjoint methodology for obtaining surface sensitivities is implemented for several equation sets within SU^2. For this problem, a new formulation based on the Euler equations in a rotating reference frame is used. After solving the direct flow problem, the adjoint problem is also solved which offers an efficient approach for calculating the gradient of an objective function with respect to a large set of design variables. This leads directly to a gradient-based optimization framework. With each design iteration, the direct and adjoint solutions are used to compute the objective function and gradient, and the optimizer drives the shape changes with this information in order to minimize the objective. Two other SU^2 tools are used to compute the gradient from the adjoint solution (SU2_GPC) and deform the computational mesh (SU2_MDC) during the process.

To run this design case, follow these steps at a terminal command line:

1. Move to the SU2/SU2Py/ directory. The shape_optimization.py script can be found here, along with the SU2_CFD (serial version), SU2_GPC, and SU2_MDC if SU^2 was built using the build_SU2.py script. If not, then compile each tool individually and make sure that a copy is placed in the SU2/SU2Py/ directory.

2. Copy the config file (rot_NACA0012.cfg) and the mesh file (mesh_NACA0012_rot.su2) to this directory.

3. Execute the shape optimization script by entering "python shape_optimization.py -f rot_NACA0012.cfg" at the command line. Again, note that Python, NumPy, and SciPy are all required to run the script.

4. The python script will drive the optimization process by executing flow solutions, adjoint solutions, gradient projection, and mesh deformation in order to drive the design toward an optimum. The optimization process will cease when certain tolerances set within the SciPy optimizer are met.

5. Solution files containing the flow and surface data will be written for each flow solution and adjoint solution and will be numbered in succession. Note that these do not correspond to major optimizer iterations, as the SciPy algorithm chosen may require several flow solutions for each iteration in the gradient-based method. The file named optimization_rot_NACA0012.csv will contain the functional values of interest resulting from each flow solution during the optimization.

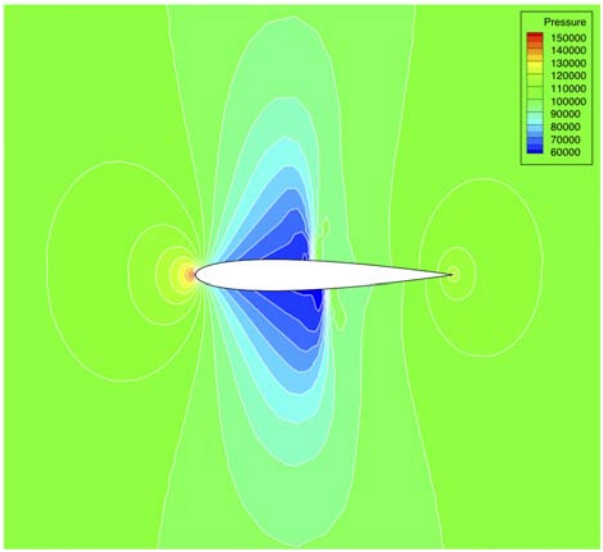


Figure 5.33: Pressure contours showing transonic shocks on the initial design.

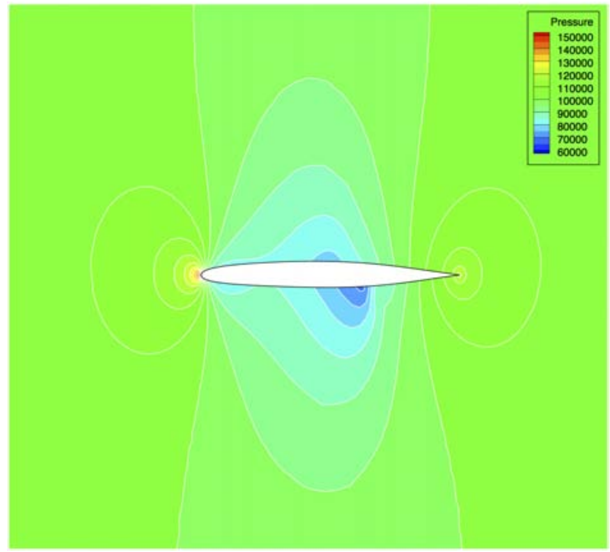


Figure 5.34: Pressure contours around the final airfoil design. Note that the shocks have essentially been removed during the design process.

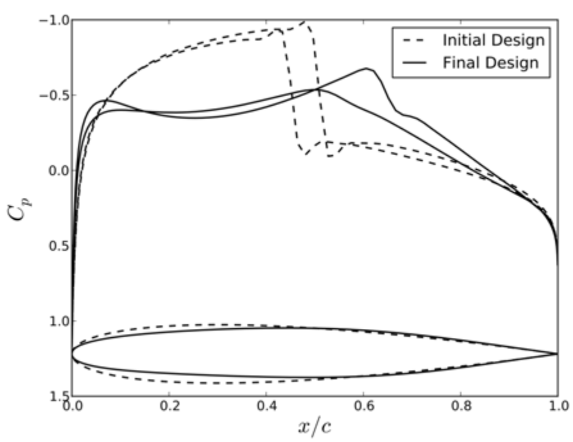


Figure 5.35: Cp distribution and profile shape comparison for the initial and final airfoil designs.

5.8.7 Results

5.9 Tutorial 9 - Optimal Shape Design of a Fixed Wing

5.9.1 Goals

Upon completing this tutorial, the user will be familiar with performing an optimal shape design of a 3-D geometry. The initial geometry chosen for the tutorial is a ONERA M6 fixed wing at transonic speed in inviscid fluid. The following SU^2 tools will be showcased in this tutorial:

- SU2_CFD - performs the direct and the adjoint flow simulations
- SU2_GPC - projects the adjoint surface sensitivities into the design space to obtain the gradient
- SU2_MDC - deforms the geometry and mesh with changes in the design variables during the shape optimization process
- shape_optimization.py - automates the entire shape design process by executing the SU^2 tools and optimizer

5.9.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/inv_ONERAM6/ directory. You will need the configuration file (inv_ONERAM6.cfg) and the mesh file (mesh_ONERAM6_inv.su2), note that the mesh file contains information about the definition of the Free Form Deformation (FFD) used for the definition of 3D design variables.

5.9.3 Tutorial

The following tutorial will walk you through the steps required when performing 3-D shape design using SU2, and FFD tools. It is assumed that you have already obtained and compiled SU2_CFD, SU2_GPC, and SU2_MDC. The design loop is driven by the shape_optimization.py script, and thus Python along with the NumPy and SciPy Python modules are required for this tutorial. If you have yet to complete these requirements, please see the Download and Installation pages.

5.9.4 Background

This example uses a 3-D fix wing geometry (initially the ONERA M6) at transonic speed in air (inviscid calculation). The design variables are defined using the FFD methodology, and at the end of the mesh_ONERAM6_inv.su2, the description of the FFD box is provided:

```
NCHUNK=1
CHUNK_TAG=0
CHUNK_DEGREE_I=5
CHUNK_DEGREE_J=4
CHUNK_DEGREE_K=1
CHUNK_CORNER_POINTS=8
-0.5 0 -0.6
8.5 0 -0.6
13 16 -0.6
8.5 16 -0.6
-0.5 0 0.6
8.5 0 0.6
13 16 0.6
8.5 16 0.6
CHUNK_CONTROL_POINTS=0
CHUNK_SURFACE_POINTS=0
```

Note that, only the corners of the box, and the polynomial degree in each direction are provided.

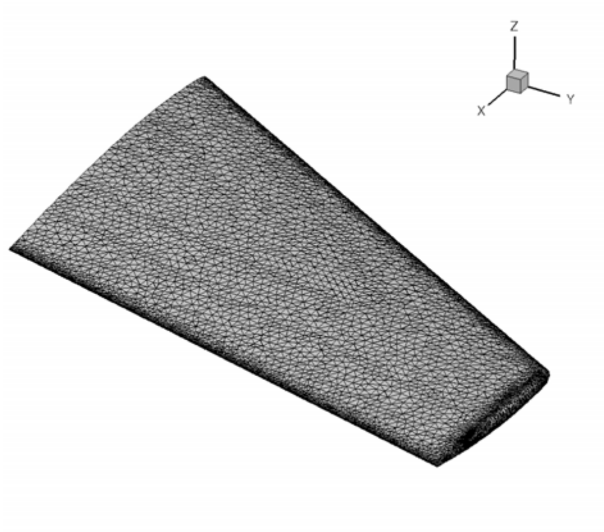


Figure 5.36: View of the initial surface computational mesh.

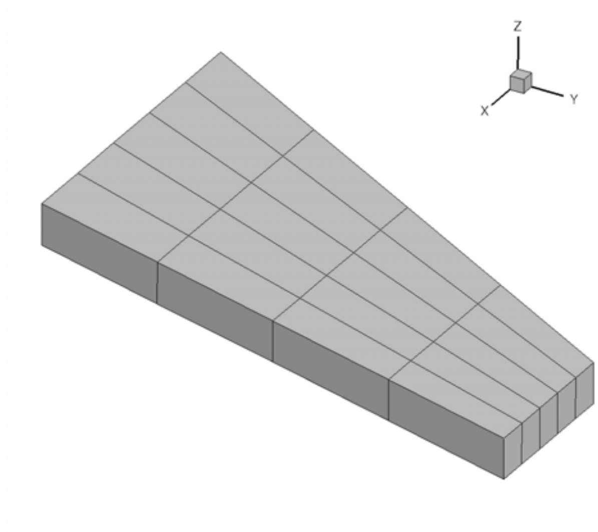


Figure 5.37: View of the initial FFD box, control points and the surface mesh.

5.9.5 Problem Setup

The goal of the design process is to minimize the coefficient of drag (Cd) by changing the shape of the airfoil without any constraints, as design variables we will use the z-coordinate of the control points position. As the shock have is located on the upper side of the wing, only the control points on the upper side will be used as design variables.

Mesh Description and Preprocessing

It consists of a far-field boundary divided in three surfaces (XNORMAL_FACES, ZNORMAL_FACES, YNORMAL_FACES), an Euler wall divided in three surfaces (UPPER_SIDE, LOWER_SIDE, TIP) and a symmetry plane (SYMMETRY_FACE). The specific wing is the ONERA M6, and more information on this simulation can be found in the configuration file. The surface mesh can be seen in Figure (1).

As we have noticed, the mesh file only contains information about the limits of the FFD box. As a preprocessing it is necessary to calculate the position of the control points and the parametric coordinates, follow these steps at a terminal command line:

1. Move to the SU2/SU2Py/ directory. The SU2_MDC can be found here if SU^2 was built using the build_SU2.py script. If not, then compile SU2_MDC individually and make sure that a copy is placed in the SU2/SU2Py/ directory.
2. Copy the config file (inv_ONERAM6.cfg) and the mesh file (mesh_ONERAM6_inv.su2) to this directory.
3. Check that DV_KIND= NO_DEFORMATION in the configuration file.
4. Execute SU2_MDC by entering "./SU2_MDC inv_ONERAM6.cfg" at the command line.
5. After a some time, a mesh file called "mesh_out.su2" is now in the directory, rename that file to "mesh_ONERAM6_inv_FFD.su2". Note that this new mesh file contains all the details of the FFD method.

With this preprocessing the position of the control points and the parametric coordinates have been calculated. The FFD box and the control points can be seen in Figure (2).

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

Simulation specification:

```
%
% Mach number (non-dimensional, based on the free-stream values)
MACH_NUMBER= 0.8395
%
% Angle of attack (degrees)
AoA= 3.06
%
% Free-stream pressure (101325.0 N/m2 by default, only for Euler equations)
FREESTREAM_PRESSURE= 101325.0
%
% Free-stream temperature (273.15K by default)
FREESTREAM_TEMPERATURE= 273.15
%
% Conversion factor for converting the grid to meters
CONVERT_TO_METER= 1.0
%
% Reference area for force coefficients (0 implies automatic calculation)
REF_AREA= 0
%
% Reference pressure (101325.0 N/m2 by default)
REF_PRESSURE= 101325.0
%
% Reference temperature (273.15 K by default)
REF_TEMPERATURE= 273.15
```

As the numerical mesh with FFD will be used in the optimization, it is fundamental to change the name of the mesh input file in the configuration file, note that we will use tecplot for the visualization.

```
% Mesh input file
MESH_FILENAME= mesh_ONERAM6_inv_FFD.su2
%
% Output file format (PARAVIEW, TECPLOT)
OUTPUT_FORMAT= TECPLOT
```

Optimal shape design specification:

```
% Objective function: (DRAG, LIFT, SIDEFORCE, PRESSURE,
% MOMENT_X, MOMENT_Y, MOMENT_Z, EFFICIENCY,
% CHARGE)
OBJFUNC= DRAG
%
% Constraint: (NONE, DRAG, LIFT, SIDEFORCE, PRESSURE,
% MOMENT_X, MOMENT_Y, MOMENT_Z, EFFICIENCY,
% CHARGE)
CONSTRAINT= NONE
%
% List of design variables (Design variables are separated by semicolons)
% - FFD_CONTROL_POINT ( 7, Scale — Mark. List — Chunk,
% i_Ind, j_Ind, k_Ind, x_Mov, y_Mov, z_Mov )
DEFINITION_DV=
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 0, 0, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 1, 0, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 2, 0, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 3, 0, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 4, 0, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 0, 1, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 1, 1, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 2, 1, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 3, 1, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 4, 1, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 0, 2, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 1, 2, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 2, 2, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 3, 2, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 4, 2, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 0, 3, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 1, 3, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 2, 3, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 3, 3, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 4, 3, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 0, 4, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 1, 4, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 2, 4, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 3, 4, 1, 0.0, 0.0, 1.0 );
( 7, 1.0 — UPPER_SIDE, LOWER_SIDE, TIP — 0, 4, 4, 1, 0.0, 0.0, 1.0 )
```

Here we define the objective function for the optimization as drag without any constraints. It is possible, for instance, to add a lift constraint. The DEFINITION_DV is the list of design variables. For this problem, we want to minimize the drag by changing the position of the control points. To do so, we define a set of FFD control points. Each design variable is separated by a semicolon. The first value in the parentheses is the variable type which is 7 for control point movement. The second value is the scale of the variable. The name between the vertical bars is the marker tag where the variable deformations will be applied. The final seven values in the parentheses are the particular information about the deformation: identification of the FFD chunk, ijk index of the control point, and xyz movement direction of the control point. Note that other types of design variables have their own specific input format.

5.9.6 Running SU^2

A continuous adjoint methodology for obtaining surface sensitivities is implemented for several equation sets within SU^2. After solving the direct flow problem, the adjoint problem is also solved which offers an efficient approach for calculating the gradient of an objective function with respect to a large set of design variables. This leads directly to a gradient-based optimization framework. With each design iteration, the direct and adjoint solutions are used to compute the objective function and gradient, and the optimizer drives the shape

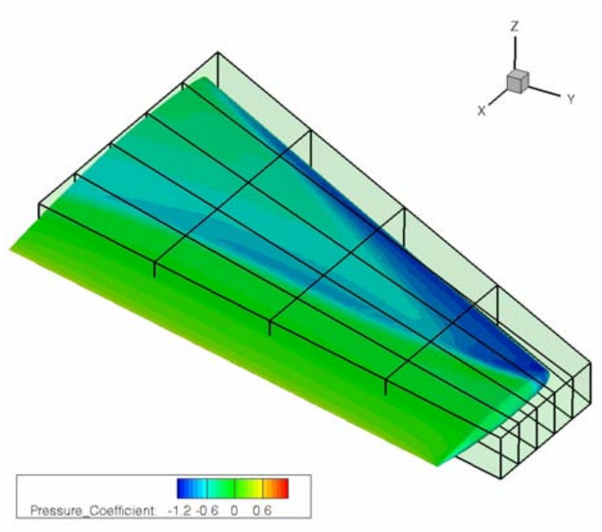


Figure 5.38: Pressure contours showing transonic shocks on the initial design.

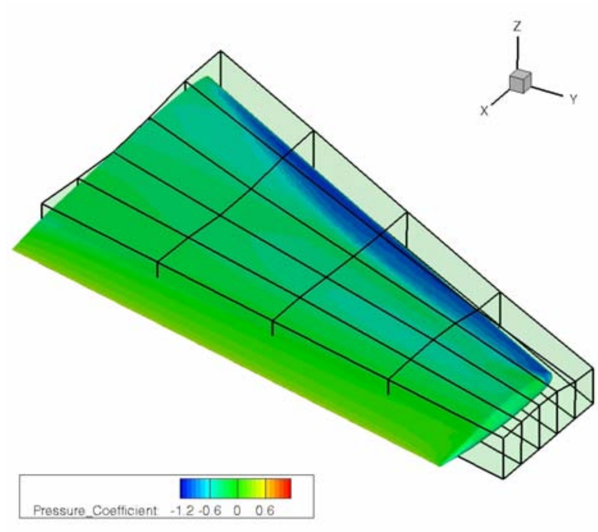


Figure 5.39: Pressure contours around the final airfoil design.

changes with this information in order to minimize the objective. Two other SU^2 tools are used to compute the gradient from the adjoint solution (SU2_GPC) and deform the computational mesh (SU2_MDC) during the process. To run this design case, follow these steps at a terminal command line:

1. Move to the SU2/SU2Py/ directory. The shape_optimization.py script can be found here, along with the SU2_CFD (serial version), SU2_GPC, and SU2_MDC if SU^2 was built using the build_SU2.py script. If not, then compile each tool individually and make sure that a copy is placed in the SU2/SU2Py/ directory.

2. Copy the config file (inv_ONERAM6.cfg) and the mesh file (mesh_ONERAM6_inv_FFD.su2) to this directory.

3. Execute the shape optimization script by entering "python shape_optimization.py -f inv_ONERAM6.cfg -g 1.0" at the command line. Again, note that Python, NumPy, and SciPy are all required to run the script.

4. The python script will drive the optimization process by executing flow solutions, adjoint solutions, gradient projection, and mesh deformation in order to drive the design toward an optimum. The optimization process will cease when certain tolerances set within the SciPy optimizer are met. Note that is is possible to start the optimization from a pre-converged solution (direct, and adjoint problem), in that case the following change should be done in the configuration file: RESTART_SOL= YES.

5. Solution files containing the flow and surface data will be written for each flow solution and adjoint solution and will be numbered in succession. Note that these do not correspond to major optimizer iterations, as the SciPy algorithm chosen may require several flow solutions for each iteration in the gradient-based method. The file named optimization_inv_ONERAM6.plt will contain the functional values of interest resulting from each flow solution during the optimization.

5.9.7 Results

5.10 Tutorial 10 - Plasma in Hypersonic Shock

5.10.1 Goals

Upon completing this tutorial, the user will be familiar with performing a multispecies simulation of plasma in the vicinity of a strong shock wave for a one dimensional flow at hypersonic speed in Argon gas. The

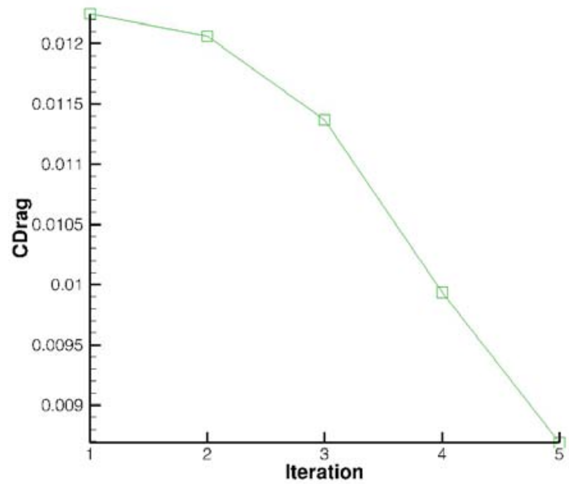


Figure 5.40: Optimization history.

solution will present the time evolution of thermal equilibrium of different species in the plasma and it will also double as a verification case for SU^2. Consequently, the following capabilities of SU^2 will be showcased and verified against published results in this tutorial:

- Unsteady, 2-D reacting-Navier Stokes equations
- Maxwell's Equations governing the electromagnetic behaviour
- Roe 1st order numerical scheme in space
- Euler Implicit time integration
- Inlet, Outlet and Symmetry boundary condition
- Navier-Stokes Wall and Farfield boundary conditions

In this tutorial, we perform our first multispecies plasma simulation with reacting- Navier Stokes equations coupled with Maxwell's equations for electrostatics.

5.10.2 Resources

The resources for this tutorial can be found in the SU2/TestCases/react_Argon/ directory. You will need the configuration file (plasma_argon_2D.cfg) and the mesh file (Argon_2D_Thin_Straight.su2).

Additionally, profiles of the density, temperature and various other thermodynamic properities of the various species in the plasma (published by MacCormack et. al, AIAA-2011-3921-256)

5.10.3 Tutorial

The following tutorial will walk you through the steps required when solving for flow through a one dimensional shock wave at Mach 15 resulting in plasma, using SU^2. It is assumed that you have already obtained and compiled the SU2_CFD code for a serial computation. If you have yet to complete these requirements, please see the Download and Installation pages.

5.10.4 Background

Formation of plasma in the vicinity of a strong shock wave is a topic of great interest for hypersonic applications. This tutorial solves for unsteady, inviscid, chemically reacting flow in the vicinity of a Mach 15 shock wave in Argon gas. The plasma is modelled as a mixture of fluids assuming continuum of species. The

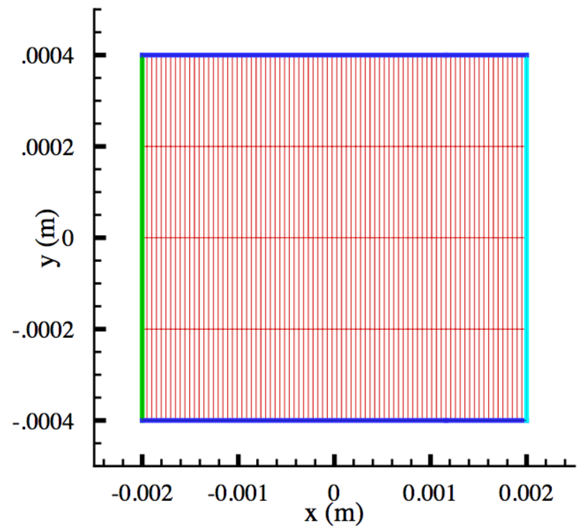


Figure 5.41: Mesh with boundary conditions (Inlet, Symmetry and Neumann).

full set of governing equations comprises of the Navier-Stokes equations governing the fluid-like behaviour of plasma, Maxwell's equations governing the electromagnetic behaviour of charged species and some relations describing the chemistry of non-equilibrium flows.

5.10.5 Problem Setup

The working gas is Argon. The initial condition is such that there is uniform flow moving at Mach 15 over half the domain, and a normal shock wave at the center of the domain. The shock wave at such high Mach number causes the temperature downstream of it to rise by several orders of magnitude, resulting in ionization of Argon gas and formation of plasma.

A coupled set of 12 Navier-Stokes equations along with relations describing the chemistry of non-equilibrium flows is solved for three species in the plasma of Argon gas, along with a solution of Maxwell's equation for electrostatics at every time step of the fluid equations. Characteristic boundary conditions are required at the inlet and symmetry boundary conditions are used at the upper and lower boundaries. Since ionization downstream of the shockwave causes the pressure of various species to rise rapidly with time, no good estimate of the back pressure can be used and a Neumann boundary condition is used at the exit.

Mesh Description

The mesh used for this tutorial is a structured mesh with 81 points along the free stream (x direction) and 5 points in the y direction. There are a total of 324 rectangular elements in the mesh which is shown below.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

For the first time in this tutorial, we will use the Multi Species Navier Stokes solver:

```
% Physical governing equations (EULER, NAVIER_STOKES,
% MULTI_SPECIES_NAVIER_STOKES)
PHYSICAL_PROBLEM= MULTI_SPECIES_NAVIER_STOKES
%
% Specify chemical model for multi-species simulations (ARGON, AIR-7)
GAS_MODEL= ARGON
%
% Convective numerical method (JST, LAX-FRIEDRICH,
% ROE-1ST_ORDER, ROE-2ND_ORDER)
CONV_NUM_METHOD_PLASMA= ROE-1ST_ORDER
%
% Source numerical method for flow equations in plasma
% (PIECEWISE_CONSTANT)
SOUR_NUM_METHOD_PLASMA= PIECEWISE_CONSTANT
%
% Number of species present in the plasma
NUMBER_OF_SPECIES_IN_PLASMA= 3
%
% Number of fluids present in the plasma
NUMBER_OF_FLUIDS_IN_PLASMA= 3
%
% Mass of each species present in the plasma in kg
PARTICLE_MASS= ( 6.63053168E-26,
6.6304405861812E-026, 9.10938188E-31 )
%
% Source numerical method for the electrostatic equation
% in plasma(PIECEWISE_CONSTANT)
SOUR_NUM_METHOD_ELEC= PIECEWISE_CONSTANT
%
% viscous numerical method for the electrostatic equation
% in plasma (PIECEWISE_CONSTANT)=
VISC_NUM_METHOD_ELEC= GALERKIN
```

These options are specific to a multiple species plasma solver. The user needs to specify the number of species present in the plasma and their molecular weights. The solver is only capable of handling Argon gas at the moment but we are working on developing a more generalized solver that can handle plasma in any gas. Euler Explicit and Euler Implicit time integration schemes are available for this solver. Roe's 1st order is the only available scheme for spatial discretization for this solver.

5.10.6 Running SU^2

This test case can be executed relatively quickly. To run it, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/ directory.
2. Copy the config file (plasma_argon_2D.cfg) and the mesh file (Argon_2D_Thin_Straight.su2) to this directory.
3. Run the executable by entering "./SU2_CFD plasma_argon_2D.cfg" at the command line.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria.
5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.10.7 Results

The figures below show the results obtained from SU^2 for plasma in vicinity of a shock wave at Mach 15

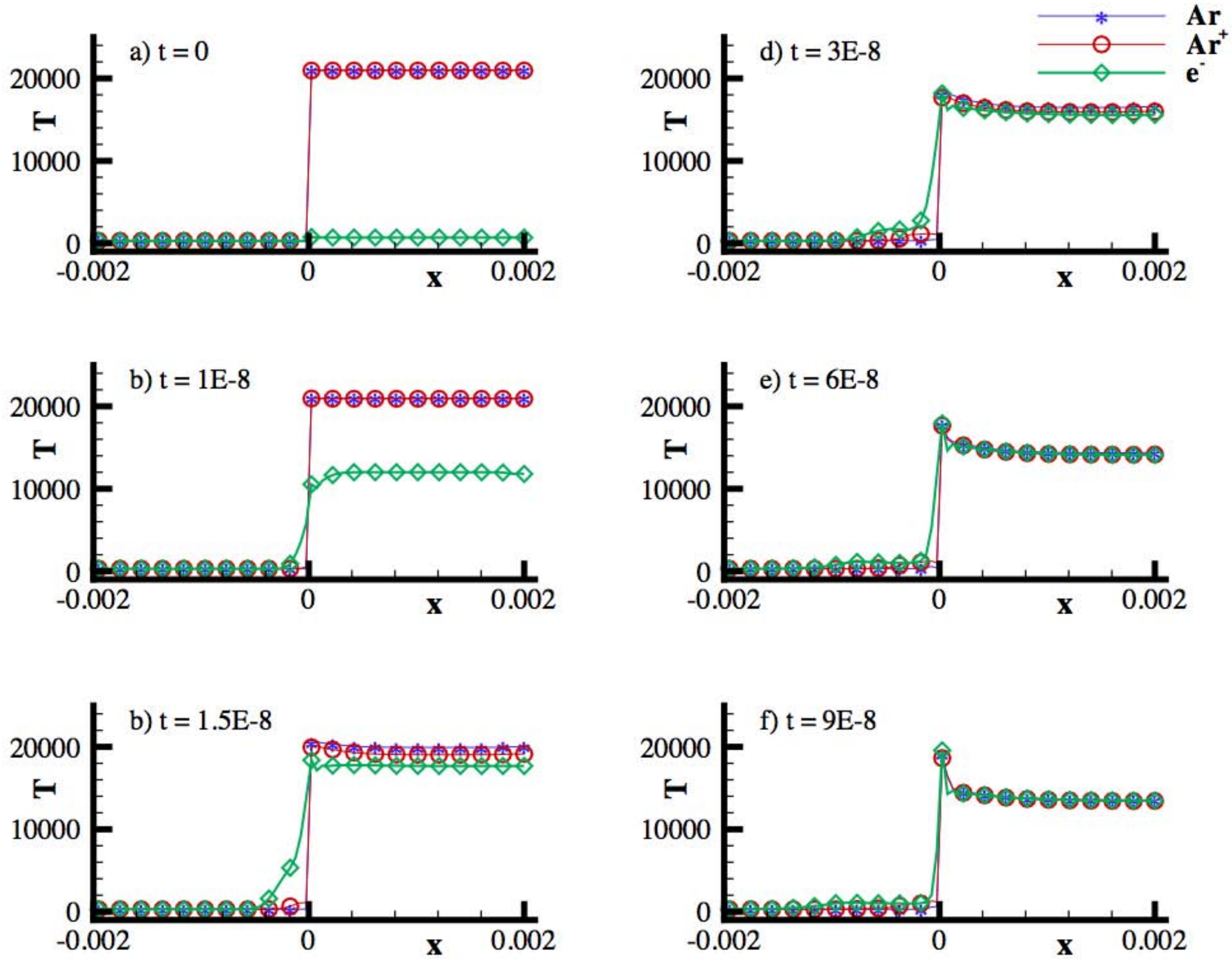


Figure 5.42: Evolution of Thermal Equilibrium in between various species with time.

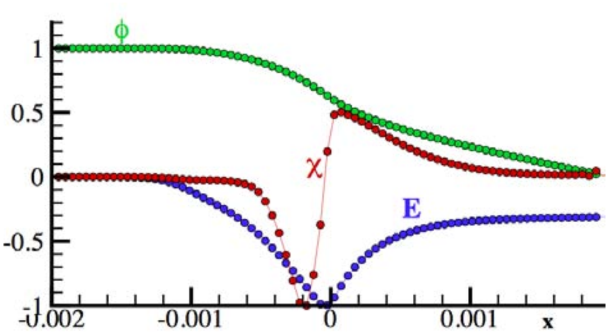


Figure 5.43: Results for Electrostatic potential (), net charge density() and Electric Field (E).

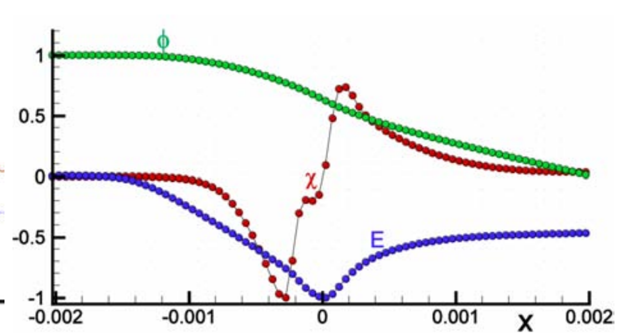


Figure 5.44: Published Results for Comparison by MacCormack et al..

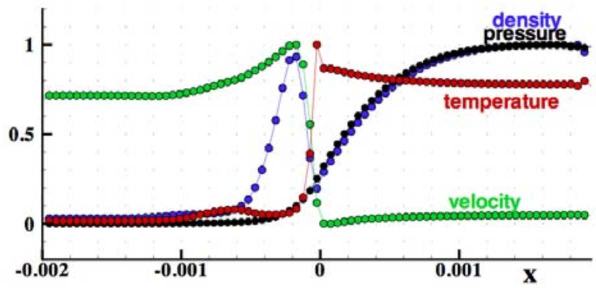


Figure 5.45: Results for the variation of thermodynamic properties of electrons.

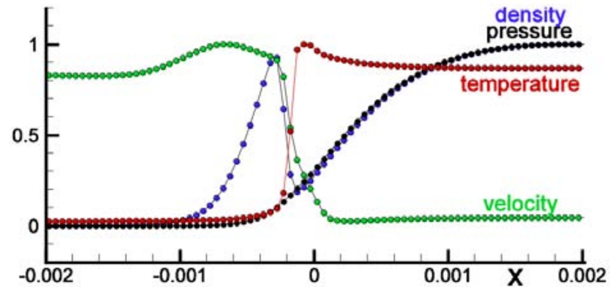


Figure 5.46: Published Results for Comparison by MacCormack et al.

5.11 Tutorial 11 - Inviscid Supersonic Wedge

5.11.1 Goals

Upon completing this tutorial, the user will be familiar with performing a simulation of supersonic, inviscid flow over a 2-D geometry. The specific geometry chosen for the tutorial is a simple wedge. Consequently, the following capabilities of SU^2 will be showcased in this tutorial:

- Steady, 2-D Euler equations
- Multigrid
- JST numerical scheme in space
- Euler implicit time integration
- Far-field, Outlet, and Euler Wall boundary conditions

The intent of this tutorial is to introduce a simple, inviscid flow problem that will allows users to become familiar with using a CGNS mesh. This will require SU^2 to be built with CGNS support, and some new options in the configuration file related to CGNS meshes will be discussed.

5.11.2 Resources

The resources for this tutorial are not currently in the TestCases/ directory. Download the mesh and configuration files here for this case:

- inv_wedge.cfg
- mesh_wedge_inv.cgns

5.11.3 Tutorial

The following tutorial will walk you through the steps required when solving for the flow using SU^2. It is assumed you have already obtained and compiled SU2_CFD with CGNS support. If you have yet to complete these requirements, please see the Download and Installation pages.

5.11.4 Background

This example uses a 2-D geometry which features a wedge along the solid lower wall. In supersonic flow, this wedge will create an oblique shock in the flow, and its properties can be predicted from the oblique-shock relations for a perfect gas (found in almost any text on compressible fluids). This is a very common test case for CFD codes due to its simplicity along with the ability to verify the code against the oblique-shock relations.

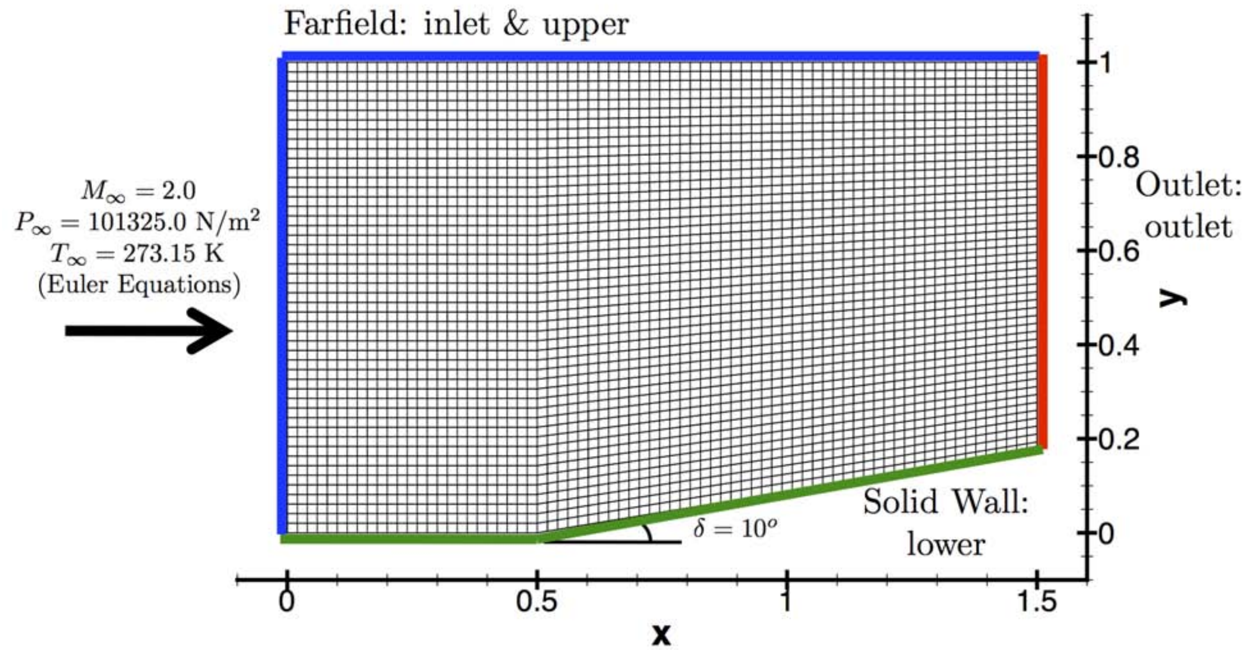


Figure 5.47: The computational mesh with boundary conditions highlighted.

5.11.5 Problem Setup

This problem will solve for the flow over the wedge with these conditions:

- Freestream Pressure = 101325.0 N/m2
- Freestream Temperature = 273.15 K
- Freestream Mach number = 2.0
- Angle of attack (AoA) = 0.0 deg

Mesh Description

The wedge mesh is a structured mesh (75x50) of rectangular elements with a total of 3,750 nodes. The lower wall of the geometry is solid and has a 10o wedge starting at x=0.5. Figure (1) shows the mesh with the boundary markers and flow conditions highlighted.

For this test case, the inlet and upper markers will be set to the far-field boundary condition, while the outlet marker will be set to the outlet condition. In supersonic flow, all characteristics point into the domain at the entrance (inlet & upper), so all flow quantities can be specified (no information travels upstream). This justifies the use of the far-field condition at these boundaries. At the exit, however, all characteristics are outgoing, meaning that no information about the exit conditions is required. Therefore, the outlet marker is set to the outlet boundary condition which, in supersonic flow, simply extrapolates the flow variables from the interior domain to the exit. In short, any back pressure can be supplied to the MARKER_OUTLET boundary condition in the configuration file, because it is ignored for this specific supersonic case.

Configuration File Options

Several of the key configuration file options for this simulation are highlighted here. Please see the configuration file page for a general description of all options.

It is recommended that you first read through the description for building and running SU^2 with CGNS support. Here we will discuss a few options in the wedge configuration file as a practical example for getting your own CGNS mesh files working:

```
%
% Mesh input file
MESH_FILENAME= mesh_wedge_inv.cgns
%
% Mesh input file format (SU2, CGNS, NETCDF_ASCII)
MESH_FORMAT= CGNS
%
% Convert a CGNS mesh to SU^2 format (YES, NO)
CGNS_TO_SU2= YES
%
% Converted SU^2 mesh filename
NEW_SU2_FILENAME= mesh_wedge_inv.su2
```

To use the supplied CGNS mesh, simply enter the filename and make sure that the MESH_FORMAT option is set to CGNS. A converter for creating .SU^2 meshes from CGNS meshes is built directly into SU^2. To use this feature, set the CGNS_TO_SU2 flag to YES and provide a name for the converted mesh (for this case, it has been set to "mesh_wedge_inv.su2"). SU^2 will convert the mesh during the pre-processing after reading in the original CGNS mesh and print the new mesh file to the current working directory. The output written to the console during this process might look like the following for the supersonic wedge mesh:

```
———————- Read grid file information ———————
CGNS mesh file format.
Reading the CGNS file: mesh_wedge_inv.cgns
CGNS file contains 1 database(s).
Database 1, Base: 1 zone(s), cell dimension of 2, physical dimension of 3.
Zone 1, dom-1: 3750 vertices, 3626 cells, 0 boundary vertices.
Reading grid coordinates...
Number of coordinate dimensions is 3.
Reading CoordinateX values from file.
Reading CoordinateY values from file.
Reading CoordinateZ values from file.
Reading connectivity information...
Number of connectivity sections is 5.
Reading section QuadElements of element type Rectangle
starting at 1 and ending at 3626.
Reading section inlet of element type Line
starting at 3627 and ending at 3675.
Reading section lower of element type Line
starting at 3676 and ending at 3749.
Reading section outlet of element type Line
starting at 3750 and ending at 3798.
Reading section upper of element type Line
starting at 3799 and ending at 3872.
Successfully closed the CGNS file.
Writing SU^2 mesh file: mesh_wedge_inv.su2.
Successfully wrote the SU^2 mesh file.
```

SU^2 prints out information on the CGNS mesh including the filename, the number of points, and the number of elements. Another useful piece of information is the listing of the zone sections within the mesh. These descriptions give the type of elements for the section as well as any name given to it. For instance, when the inlet boundary information is read, SU^2 prints "Reading section inlet of element type Line" to the console. This information can be used to verify that your mesh is being read correctly, or even to help you remember, or learn for the first time, the names for each of the boundary markers. Lastly, if conversion to the .SU^2 format is requested, SU^2 will communicate whether the .SU^2 mesh was successfully written.

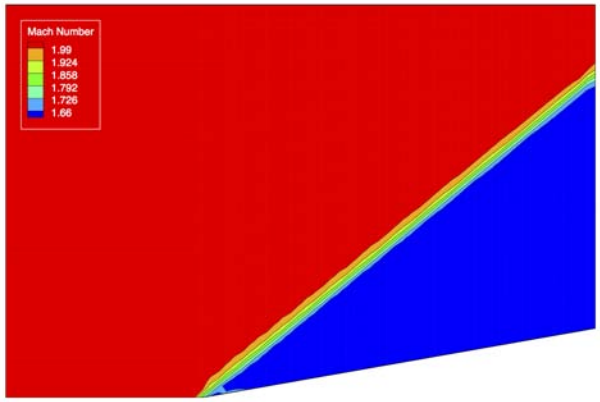


Figure 5.48: Mach contours showing the oblique shock for supersonic flow over a wedge.

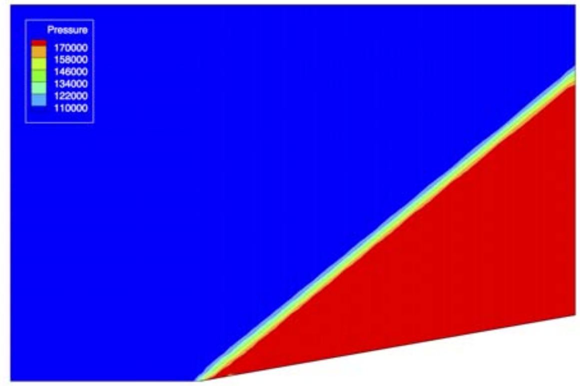


Figure 5.49: Pressure contours (N/m2) for supersonic flow over a wedge.

5.11.6 Running SU^2

The wedge simulation is small and will execute quickly on a single workstation or laptop, and this case will be run in a serial fashion. To run this test case, follow these steps at a terminal command line:

1. Move to the directory containing the compiled executable of SU2_CFD (serial version). If you built the code with the build_SU2.py script, SU2_CFD can be found in the SU2/SU2Py/directory.
2. Copy the config file (inv_wedge.cfg) and the mesh file (mesh_wedge_inv.cgns) to this directory.
3. Run the executable by entering "./SU2_CFD inv_wedge.cfg" at the command line.
4. SU^2 will print residual updates with each iteration of the flow solver, and the simulation will finish after reaching the specified convergence criteria.
5. Files containing the results will be written upon exiting SU^2. The flow solution can be visualized in ParaView (.vtk) or Tecplot (.plt).

5.11.7 Results

The following images show some SU^2 results for the supersonic wedge problem.

Chapter 6

Developer's Guide

This Developer's Guide includes an overview of the SU^2 project, guidelines for anyone who wants to help develop the code, and information on contributing to this Wiki.

The full technical documentation of the code, explaining the structure and details of the source code for developers is generated through Doxygen and available here.

For information about how to use the suite please take a look at the resources in the User's Guide.

6.1 Open source philosophy

The source code of open source software is made freely available, though usually under some sort of license, so that anyone may copy, edit and distribute it without needing to pay for these rights. The intention of this is that it evolves and is improved upon through the cooperation of its user and developer community. The Open Source Initiative provides this detailed definition that better explain exactly what this means.

Additional information on the open source philosophy and open source software can also be found on Wikipedia.

6.2 Coding style guidelines

SU^2 is released under an open source license to facilitate its widespread use and development in the scientific computing community. To support uniformity and consistency in the style of the source code, a C++ style guide has been included on this page, and it is strongly encouraged that outside developers adhere to the guidelines dictated in the style guide to maintain readability of the source.

A discussion of the open source philosophy adopted by the ADL development team and its impact on the evolution of the software tool is provided here.

Any contributions from the scientific community at-large are encouraged and welcomed. Feel free to contact the development team at any time.

This document describes the conventions that will be used when implementing new features in SU^2. This includes allowed syntactic and semantic language features, filename conventions, indentation conventions and more. The consistency is fundamental, it is very important that any programmer be able to look at another part of the code and quickly understand it, the uniformity in the style is a key issue. Some of the ideas expressed in this document comes from the Google C++ Style Guide (revision 3.188).

6.2.1 C++ style guide

Version numbering

Each code of the SU^2 suite must have a release number following the rule Major.Patch, where the Major number is increased each time a new major update is performed and the Patch number is increased each time new features are added. The configuration file also has a number following the rule Major.Patch, where Major correspond with the SU2_CFD major version and Patch is increased with new changes.

Standard conformance, and formatting

Source code must comply with the C++ ISO/ANSI standard. with respect to the formatting some recommendation can be made:

- Each line of text in your code should be at most 80 characters long.
- Non-ASCII characters should be rare, and must use UTF-8 formatting.
- Use only tabs. You can set your editor to emit spaces when you hit the tab key.
- Sections in public, protected and private order, each indented one space.
- The hash mark that starts a preprocessor directive should always be at the beginning of the line.
- When you have a boolean expression that is longer than the standard line length, be consistent in how you break up the lines.

Files, functions, and variables

Here some basic recommendation are made for creating files, functions, and variables:

- C++ filenames must have extension .cpp.
- C++ header filenames must have extension .hpp. In general, every .cpp file should have an associated .hpp file.
- C++ inline filenames must have extension .inl. Define functions inline only when they are small, say, 10 lines or less.
- All subprograms (subroutines of functions) must be contained in a class. Each parent class must be contained in a file with the same name as the class (plus extension .cpp, and .hpp).
- This implies that there can only be one parent class per file.
- When defining a function, parameter order is: inputs, then outputs.
- Order of includes. Use standard order for readability and to avoid hidden dependencies: C library, C++ library, other libraries', your project's.
- Prefer small and focused functions.
- Use overloaded functions (including constructors) only if a reader looking at a call site can get a good idea of what is happening without having to first figure out exactly which overload is being called.
- Local variables. Place a function's variables in the narrowest scope possible, and initialize variables in the declaration.
- Static or global variables of class type are forbidden: they cause hard-to-find bugs due to indeterminate order of construction and destruction.
- In the initialization, use 0 for integers, 0.0 for reals, NULL for pointers, and '\ 0' for chars.

Classes

The classes are the key element of the object oriented programming, here some basic rules are specified.

In general, constructors should merely set member variables to their initial values. Any complex initialization should go in an explicit Init() method.

- You must define a default constructor, and destructor.
- Use the C++ keyword explicit for constructors with one argument.
- Use a struct only for passive objects that carry data; everything else is a class.
- Do not overload operators except in rare, special circumstances.
- Use the specified order of declarations within a class: public: before private:, methods before data members (variables), etc.

Syntactic and semantic requirements

In this section you can find some basic rules for programming:

- All allocated memory must be deallocated at program termination.
- Read or write operations outside an allocated memory block are not allowed.
- Read or write outside index bounds in arrays or character variables are not allowed.
- No uninitialized/undefined values may be used in a way that could affect the execution.
- Local variables that are not used must be removed.
- Pointer variables must be initialized with NULL unless they are obviously initialized in some other way before they are used.
- Indentation will be two steps for each nested block-level.
- In the header file, at the beginning of each program unit (class, subroutine or function) there must be a comment header describing the purpose of this code. The doxygen format should be used.
- When possible, it is better to use #DEFINE with a physical meaning to simplify the code.
- The code must be compiled using doxygen to be sure that there is no warning in the commenting format.
- When describing a function the following tag must be used: \brie_, \para_\[in\], \para_\[out\], \retur_, \overload.
- Static or global variables of class type are forbidden: they cause hard-to-find bugs due to indeterminate order of construction and destructionUse 0 for integers, 0.0 for reals, NULL for pointers, and '\0' for chars.
- All parameters passed by reference must be labeled const. We strongly recommend that you use const whenever it makes sense to do so.
- In the code short, int, and the unsigned version of both must be used case depending. Code should be 64-bit and 32-bit friendly. Bear in mind problems of printing, comparisons, and structure alignment.

Naming

The most important consistency rules are those that govern naming. The style of a name immediately informs us what sort of thing the named entity is: a type, a variable, a function, a constant, a macro, etc., without requiring us to search for the declaration of that entity. Here you can find some basic rules:

The following naming conventions for variables must be used:

- Geometry: Normal, Area (2D, and 3D), Volume (2D, and 3D), Coord, Position. Solution: Solution, Residual, Jacobian.
- Function names, variable names, and filenames should be descriptive; eschew abbreviation.
- Types and variables should be nouns, while functions should be "command" verbs.
- Elementary functions that set or get the value of a variable (e.g. Number) must be called as
- GetNumber(), or GetNumber(). Function names start with a capital letter and have a capital letter for each new word, with no underscores.
- Variable names are all lowercase, with underscores between words.
- The name for all the classes must start with the capital "C" letter, followed by the name of the class (capitalizing the first letter), if the name is composed by several words, all the words must be together, e.g.: CPrimalGrid.
- All the variables that are defined in a class must be commented using /*¡ \brie _________.*/.

Comments

The documentation, and comments must be Doxygen friendly, here I include some basic features:

- Start each file with a copyright notice, followed by a description of the contents of the file.
- Every class definition should have an accompanying comment that describes what it is for and how it should be used.
- Declaration comments describe use of the function; comments at the definition of a function describe operation. In general the actual name of the variable should be descriptive enough to give a good idea of what the variable is used for.
- In your implementation you should have comments in tricky, non-obvious, interesting, or important parts of your code.
- Pay attention to punctuation, spelling, and grammar; it is easier to read well-written comments than badly written ones.
- Short, and long comments must be in inside of /*\-**-\- –*\-/, and they must be located just before the line to be commented.
- Math comments are welcome and should be in the Latex language.

Debugger tools

The C++ code must support the following features for debugging:

- Array index bounds may be checked at runtime.
- Conformance with C++ may be checked.
- Use of obsolescent features may be reported as compilation warnings.
- Unused variables may be reported as compilation warnings.
- Iteration: iPoint, jPoint, kPoint, iNode, jNode, kNode, iElem, jElem, kElem, iDim, iVar, iMesh, iEdge.

6.3 SU^2 Documentation Wiki

Like the actual SU^2 code itself, this living documentation is designed so that many people can contribute to and improve it. This website is a Wiki based on the Confluence software package developed by Atlassian. Currently, the ability to edit this documentation is limited to the SU^2 development team. If you are interested in joining the development team, please contact us.

Some very basic guidelines to help keep the style reasonably consistent can be found here, and recent changes to the site are listed here.

6.3.1 Wiki style guide

The following is intended to be a simple list of rules that will hopefully keep the documentation on this Wiki reasonably uniform. For a more general idea of the style of the site it is suggested simply to take a look at existing pages. One of the main principles of this site is that pages need to be kept relatively clear and concise, and that the general structure of the documentation should be simple and allow users to navigate the site easily.

Heading Hierarchy and Colors

- The headings on a page should try to follow the order: Page Title, Heading 1, Heading 2, Heading 3, etc. This means, for example, that the first heading after the Page Title, should be Heading 1. In cases where there are too many headings on a particular page, the rule can (and should) be broken for cosmetic reasons and a smaller heading size can be chosen.
- Normal Paragraph text can be placed at any point within this hierarchy, and the lowest level of text on the page must be Paragraph style. For example, Paragraph text can optionally be placed between Page Title and Heading 1, and if Heading 3 is the last heading level it should be followed by Paragraph, not another heading style.
- The Table of Contents macro can be used at the top of the page to display a list of all headings on the current page.
- Headings 1 and 2 should be colored red, and lower headings black.

Text and Writing Styles

- Use bold text to highlight words and where list items have titles.
- Use italics to indicate code or commands. Additionally the Code Block macro can be used (see below).
- The 2 in SU^2 should be a superscript.
- All sentences, including at the end of each paragraph must end in a period '.'. Also the spacing between sentences should be a single white space.
- All acronyms should be explained the first time they are used.

Bullets and Numbering

- List items that begin with a title followed by text on the same line should have the title in bold followed by a hyphen (Title - Text).
- List items that begin with a title followed by test on the next line should have the title in bold followed by a colon (Title:).
- List items without titles and the main body of list items with titles should begin in regular text.
- Bulleted, numbered and simply indented lists can be used interchangeably (even within the same list) except that a list detailing a specific sequence of actions must be numbered.

Figures

Captions should be below the pictures and of the format: Figure (1): Caption.

Useful Macros

If you start by typing an opening brace (aka curly bracket) followed by some text Confluence will find a list of macros matching that text. The following may be useful:

- Children Display - Lists the child pages of the current page.
- Code Block - Marks a section of code.
- Column - Breaks the page into vertical columns.
- Page Tree - Shows the child pages and hierarchy below a page.
- Section - Breaks the page into horizontal sections.
- Table of Contents - Insert a table of contents of all the headings in the current page.

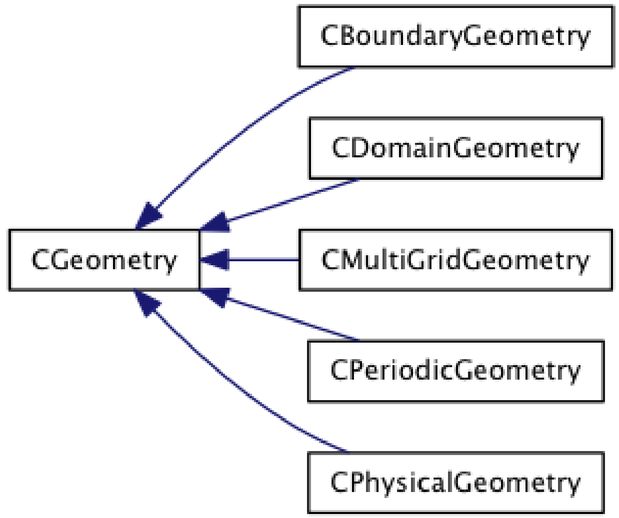


Figure 6.1: Hierarchy of CGeometry class.

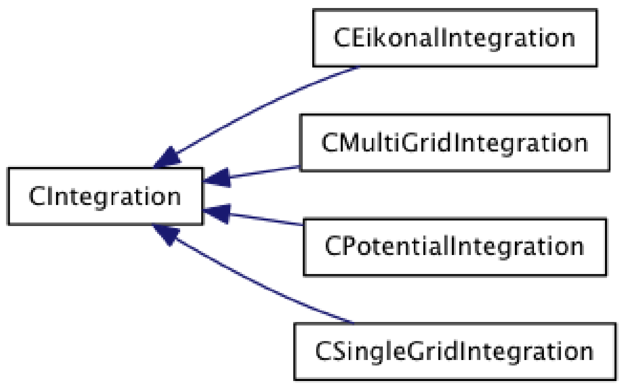


Figure 6.2: Hierarchy of CIntegration class.

6.4 Code Structure

Full details on the class hierarchy and internal structure of the code can be found in the Doxygen documentation for SU^2. A brief description for each the major C++ classes is given on this page.

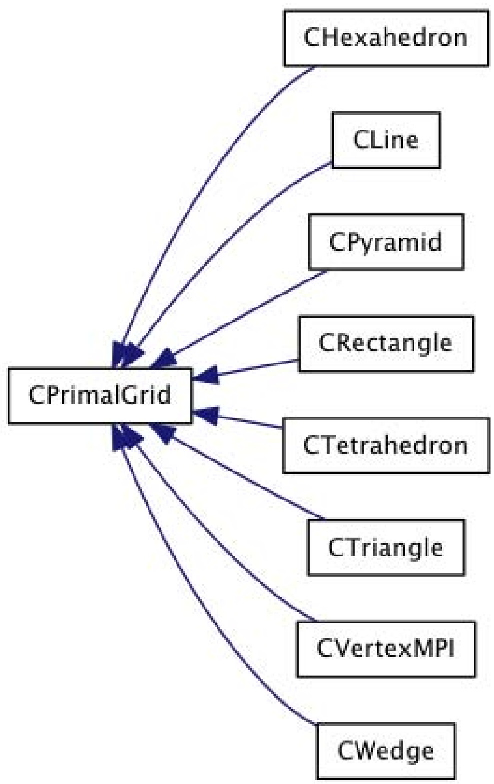


Figure 6.3: Hierarchy of CPrimalGrid class.

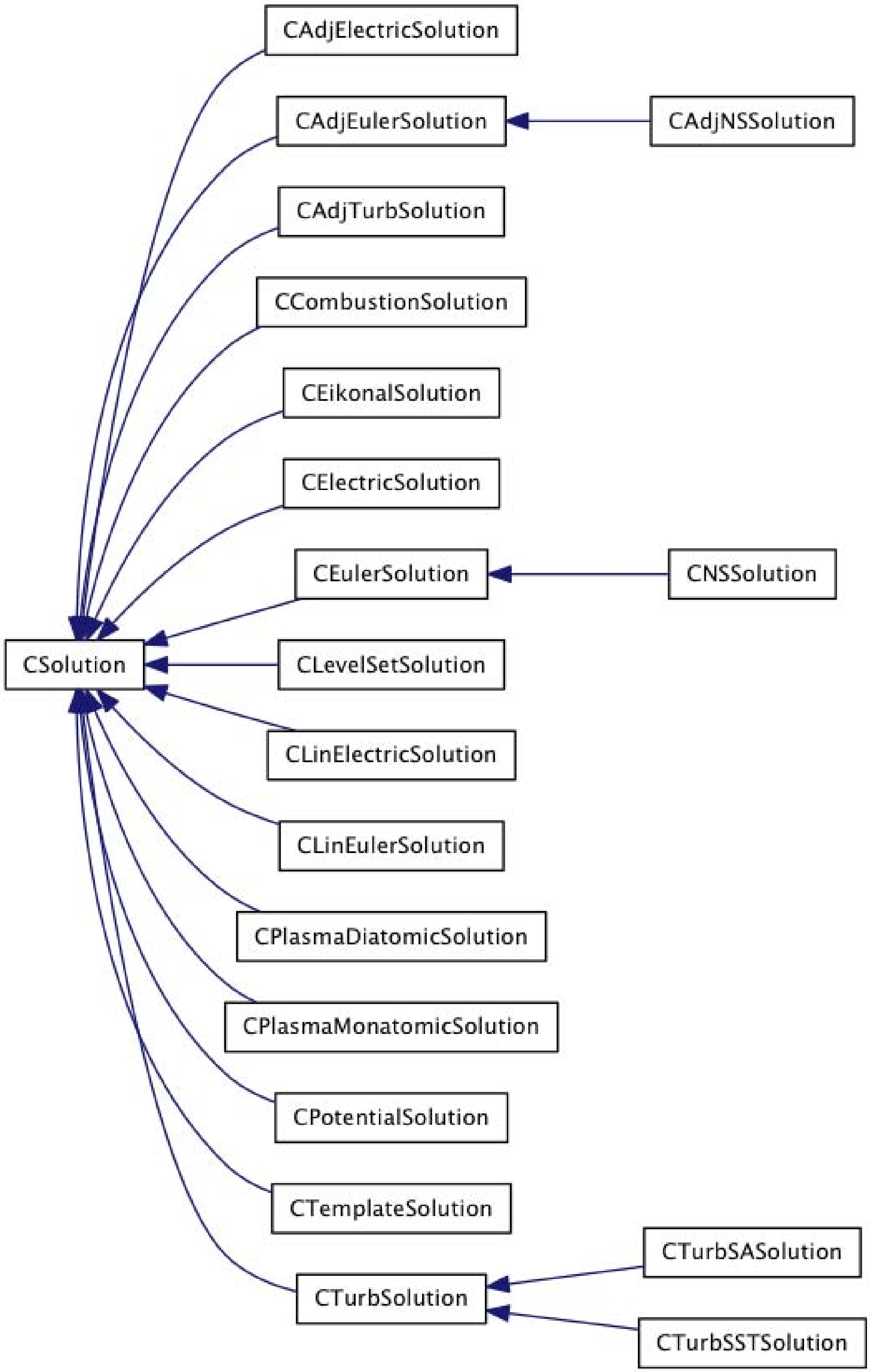


Figure 6.4: Hierarchy of CSolution class.

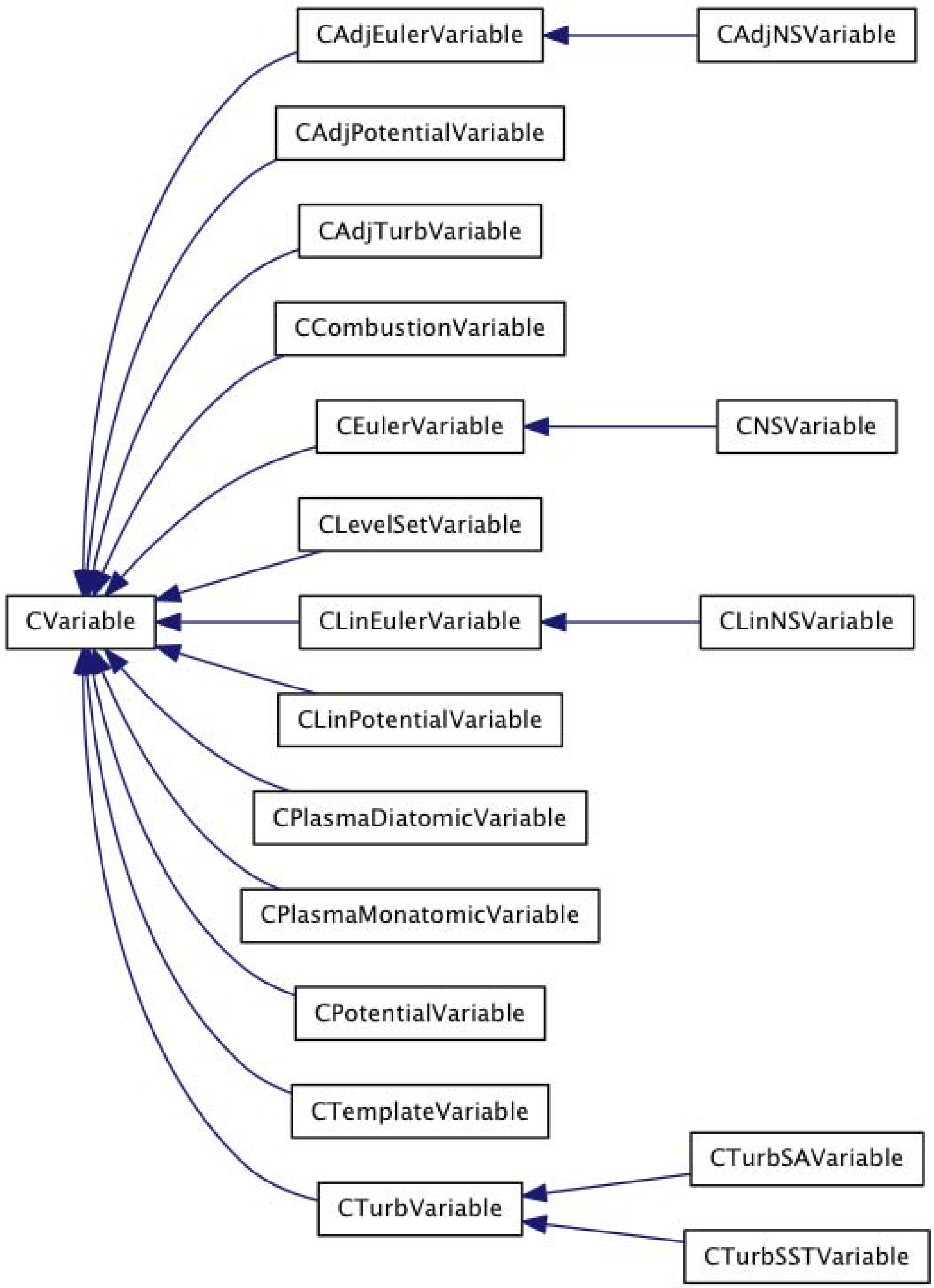


Figure 6.5: Hierarchy of CVariable class.

Appendix A

Frequently Asked Questions

For additional help or information about features of the SU^2 code not explicitly provided in the User's Guide or Developer's Guide please check out the answers below. If you still need help, you can contact us here.

A.0.1 Can I use SU^2 for my own research?

Absolutely! Our goal is to develop and maintain the premier code in the world for PDE analysis and design on unstructured meshes, and a strong user and developer base is vital to the continued growth and development of the SU^2 platform. That is why we are freely releasing the suite to the general public under an open source license. Please give our code a try, send any feedback to the developers, and recommend it to your colleagues if you are impressed with the performance. Please read the license details here.

A.0.2 Where can I download SU^2?

See the Download page to obtain either the source code or precompiled binary executables for select platforms. Note that SU^2 is released under an open source license.

A.0.3 How do I contact the developers?

The developers can be contacted through the developer's mailing list. Send an email directly to the developer's list if you have any feedback, such as bug reports, feature requests, or problems with the code. Your useful questions may end up here in the FAQs section. Users are also encouraged to join the user's mailing list in order to receive important updates on new releases or bug fixes. Instructions for using both mailing lists are found on the Contact page.

A.0.4 How do I report a bug?

Send an email directly to the developer's mailing list address given on the Contact page.

A.0.5 What types of computational meshes does SU^2 support?

SU^2 supports both a native format (.su2) and the CGNS data standard (.cgns) for mesh input. CGNS support is currently limited to mesh input for serial simulations only, but both parallel capability and solution output will be added in the future. Please see the page on meshes for detailed descriptions of how to create and use these two formats with SU2.

A.0.6 Which visualization packages can I use to view my solutions?

SU^2 outputs solution files in either VTK format for viewing in ParaView or Tecplot format. More information on obtaining these two packages can be found on the installation page.

A.0.7 Where I can get the Metis partitioner?

The Metis partitioner can be found in http://glaros.dtc.umn.edu/gkhome/fsroot/sw/metis/OLD. Currently 4.0.3 is the version supported by the build script.

A.0.8 How do I generate some simple plots using ParaView?

Paraview is available for free download here. The following tips are intended to be used in conjunction with the Quick Start Tutorial, so please follow the steps on that page to generate the necessary solutions for the tips provided below.

3D Contour Plots

After running the solutions in the Quick Start Tutorial, the files flow.vtk and adjoint.vtk should be located in your current working directory (note that although we solved a 2D problem these are treated as 3D because the z-coordinate is defined).

To plot the flow:

- Start Paraview.
- Open the file flow.vtk
- In the Object Inspector (bottom left of the screen) click Apply under the Properties tab.
- To fine-tune the picture:
  - Zoom in using the mouse or with the Zoom to Box tool in the menu bar.
  - Turn on the legend by clicking the icon on the menu bar named Toggle Color Legend Visibility .
  - Change the color scheme by clicking the icon on the menu bar named Color Scale Editor. In the new window under the Color Scale tab click Choose Preset, and in the following window select Blue to Red Rainbow.
  - Rescale the data range automatically using the Rescale to Data Range icon.
  - Choose the variable to be plotted in the drop-down box just to the right of the Rescale icon.

Line-charts

In this tutorial these include the files surface_flow.csv, history_flow.csv, surface_adjoint.csv and history_adjoint.csv.

To plot the surface flow:

- Start Paraview.
- Open the file surface_flow.csv
- In the Object Inspector click Apply under the Properties tab. A table will open in the main viewing window.
- Close the table by clicking the X on the top right of the viewing window, and in the Create View menu that opens select Line Chart View.
- In the Object Inspector, under the Display tab, enable Visible.
- To fine-tune the picture:
  - In the Display tab:
    - Under X Axis Data select Use Data Array and ensure it is using the x_coord
    - Under Line Series deselect everything except Pressure_Coefficient, and click on the Legend Name to change this variable name to Cp
  - Click the small Edit View Options icon that is on the left just above the line-chart to open the View Settings window:
    - Change the Chart Title to 'Cp variation on the NACA0012 airfoil' with a font size of 18
    - Turn off the Chart Legend

A.0.9 Where can I find the test case files?

The files required for the test cases detailed in the user tutorials are all included in the source code tar file on the Download page. More specifically, they can be found in the SU2/TestCases/ directory.

A.0.10 SU^2 won't compile, what should I do?

Detailed information on compilation can be found on the installation page. Automated compilation with the build_SU2.py script is recommended, if Python is available on your machine. If you are building individual SU^2 components using the makefiles, do not forget to set the SU2_HOME environment variable (SU2_HOME=/path/to/SU2) in your shell. Also check that you have a working C++ compiler, and note that the GNU and Intel C++ compilers are currently the only tested compilers by the developers. If you continue having difficulty or receive repeated compiler errors, please contact the developers.

A.0.11 I am having trouble with the Python scripts, what can I do?

First, see the installation page and make sure that you have a working version of Python (Version 2.6 recommended) on your machine. In limited situations, the scripts may require external packages (namely NumPy and SciPy) which can also be freely obtained. If you are stuck, remember that the options for a specific script can be viewed by entering "python script_name.py -h" at the command line in order to see the help menu. It is important to provide the Python scripts with the correct number of inputs with the correct syntax.

A.0.12 The SU2_CFD code compiles without problems, but the code doesn't work, what should I do?

This problem has been detected using MVAPICH2-1.7, and Microsoft Visual Studio 8.0. In short, some pointers were assumed to be initialized to NULL, which is not required by the c++ standard. This oversight has been corrected. Please copy the following files config_structure.hpp, and option_structure.hpp to the Common/include/ folder, and solution_direct_mean.cpp to the SU2_CFD/src/ folder, and recompile the code. Or, simply download a "fresh" copy of the software to resolve the problem.

A.0.13 What is the convention for the freestream flow direction?

SU^2 assumes a particular orientation for the computational mesh in 2-D or 3-D space. By convention, for zero angle of attack in a 2-D domain, the freestream is in the direction in the positive x-axis. By adjusting the angle of attack, users will control the component of freestream in the y-direction. For 3-D, the assumed freestream direction remains along the positive x-axis. However, the angle attack will control the flow direction in the x-z plane, while the sideslip angle will control the flow direction in the x-y plane.

A.0.14 How to install SU^2 in the Windows operating system without using Cygwin?

The following steps can be used to install the SU^2 CFD tool in a Windows machine. First, download the gcc compiler for windows at the following website: http://www.equation.com/servlet/equation.cmd?fa=fortran. Second, define the environment variable SU2_HOME at the beginning of your makefile that lives in the SU2_CFD directory (remember that the variable SU2_HOME should define the working directory, which is /SU2v1.0). Third and last, in the DOS terminal, go to the SU2_CFD directory and execute "make all" to get the executable files.

A.0.15 How do I run my simulation for a specific number of iterations?

Rather than use the RESIDUAL or CAUCHY options for reaching a certain level of convergence, users can input an integer number of iterations for the solver to perform in the EXT_ITER option in the config file. The simulation will terminate after reaching the specified iteration number. None Edit Labels

Appendix B

References

The following reports, articles, and texts were used as reference material during the implementation of the SU^2 suite. Please see these references for more detail on particular components of the code:

- Barth, T., Numerical Methods for Conservative Laws on Structured and Unstructured Meshes, VKI March 2003 Lecture Series.
- Hirsch, C., Numerical Computation of Internal and External Flows, Volume 2, John Wiley and Sons, 1990.
- Jameson, A., Analysis and Design of Numerical Schemes for Gas Dynamics 1 Artificial Diffusion, Upwind Biasing, Limiters and Their Effect on Accuracy and Multigrid Convergence, RIACS Technical Report 94.15, International Journal of Computational Fluid Dynamics, Vol. 4, 1995, pp. 171-218.
- Jameson, A., Analysis and Design of Numerical Schemes for Gas Dynamics 2 Artificial Diffusion and Discrete Shock Structure, RIACS Report No. 94.16, International Journal of Computational Fluid Dynamics, Vol. 5, 1995, pp. 1-38.
- Jameson, A., The present status, challenges, and future developments in computational fluid dynamics, AGARD, editor, Progress and challenges in CFD methods and algorithms, 1995.
- LeVeque, R. J., Numerical Methods for Conservation Laws, Birkhauser Verlag, 2006, Lectures in Mathematics.
- Mavriplis, D., On convergence Acceleration Techniques for Unstructured Meshes, ICASE Report No, 98-44.
- Quarteroni, A. and Valli, A., Numerical approximation of partial differential equations, Springer Series in Computational Mathematics. Springer, 1997.
- Sethian, J., Level Set Methods and Fast Marching Methods, Cambridge University Press, 1999.
- Toro, E. F., Riemann Solvers and Numerical Methods for Fluid Dynamics: a Practical Introduction, Springer-Verlag, 1999.
- Wesseling, P., Principles of Computational Fluid Dynamics, Springer Series in Computation Mathematics, Springer, 2000.
- R. W. MacCormack and D. D'Ambrosio and D.Giordano and J. K. Lee and T. Kim, Plasmadynamic Simulations with Strong Shock Waves, AIAA, 2011.
- M. I. Hoffert and H. Lien, Quasi-One Dimensional Nonequilibrium Gas Dynamics of Partially Ionized Two-Temperature Argon, Physics of Fluids, 1967.
- Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," Recherche Aerospatiale, No. 1, 1994, pp. 5-21.

List of Figures