51st AIAA Aerospace Sciences Meeting including the New Horizons Forum and Aerospace Exposition
07 - 10 January 2013, Grapevine (Dallas/Ft. Worth Region), Texas

AIAA 2013-0287

Stanford University Unstructured (SU^2): An open-source integrated computational environment for multi-physics simulation and design.

Francisco Palacios[*], Michael R. Colonno[*],
Aniket C. Aranake[†], Alejandro Campos[†], Sean R. Copeland[†], Thomas D. Economon[†],
Amrita K. Lonkar[†], Trent W. Lukaczyk[†], Thomas W. R. Taylor[†],
and Juan J. Alonso[‡]

Stanford University, Stanford, CA 94305, U.S.A.

This paper describes the history, objectives, structure, and current capabilities of the Stanford University Unstructured (SU^2) tool suite. This computational analysis and design software collection is being developed to solve complex, multi-physics analysis and optimization tasks using arbitrary unstructured meshes, and it has been designed so that it is easily extensible for the solution of Partial Differential Equation-based (PDE) problems not directly envisioned by the authors. At its core, SU^2 is an open-source collection of C++ software tools to discretize and solve problems described by PDEs and is able to solve PDE-constrained optimization problems, including optimal shape design. Although the toolset has been designed with Computational Fluid Dynamics (CFD) and aerodynamic shape optimization in mind, it has also been extended to treat other sets of governing equations including potential flow, electrodynamics, chemically reacting flows, and several others.

In our experience, capabilities for computational analysis and optimization have improved considerably over the past two decades. However, the ability to integrate the resulting software packages into coupled multi-physics analysis and design optimization solvers has remained a challenge: the variety of approaches chosen for the independent components of the overall problem (flow solvers, adjoint solvers, optimizers, shape parameterization, shape deformation, mesh adaption, mesh deformation, etc) make it difficult to (a) expand the range of applicability to situations not originally envisioned, and (b) to reduce the overall burden of creating integrated applications. By leveraging well-established object-oriented software architectures (using C++) and by enabling a common interface for all the necessary components, SU^2 is able to remove these barriers for both the beginner and the seasoned analyst.

In this paper we attempt to describe our efforts to develop SU^2 as an integrated platform. In some senses, the paper can also be used as a software reference manual for those who might be interested in modifying it to suit their own needs. We carefully describe the C++ framework and object hierarchy, the sets of equations that can be currently modeled by SU^2, the available choices for numerical discretization, and conclude with a set of relevant validation and verification test cases that are included with the SU^2 distribution. We intend for SU^2 to remain open source and to serve as a starting point for new capabilities not included in SU^2 today, that will hopefully be contributed by users in both academic and industrial environments.

[*]Engineering Research Associate, Department of Aeronautics & Astronautics, AIAA Member.
[†]Ph.D. Candidates (authors in alphabetical order), Department of Aeronautics & Astronautics, AIAA Student Members.
[‡]Associate Professor, Department of Aeronautics & Astronautics, AIAA Associate Fellow.

I. Introduction

The solution of challenging multi-disciplinary problems involves the interaction between separate modules that represent different physical models and procedures, and for avoiding efficiency and integration issues, it can be beneficial for the modules to share a common set of numerical solution algorithms and code architecture. For example, the unsteady aero-acoustic design of plasma actuators for wind turbines requires the solution of the plasma equations around the actuator, the Reynolds-averaged Navier-Stokes (RANS) equations in the neighborhood of the turbine, the Ffowcs Williams and Hawkings model for far-field noise propagation, and the entire system might need to be coupled with a Finite Element Method (FEM) model of the blade structure. Additionally, adjoint equations might need to be solved for each system, and the entire problem may require an infrastructure for shape design (shape parameterization, grid deformation, and optimization algorithms) to be available. In many other situations of aerospace interest (supersonic low-boom design, helicopter rotor optimization, uncertainty quantification in re-entry vehicles, hypersonic propulsion systems, advanced environmentally-friendly aircraft, modern engine designs, etc.) the situation is similar: the ability to integrate multiple physics, parameterize the resulting system, and design/optimize the outcome is of utmost importance and must be accomplished with minimum effort.

Although it is possible to identify the key characteristics of computational analysis and design / optimization suites that lead to capabilities and efficiencies mentioned above, one rarely has the opportunity and the resources to create such environments from the ground up. As a consequence, the architecture of the resulting efforts lacks the necessary flexibility and sophistication to overcome all the challenges. Fortunately, SU^2 was developed from scratch to overcome some of these limitations. In addition, the modern high-level programming languages leveraged in SU^2 (C++ in our case) have long provided the capabilities to ensure portability, code reuse, and the flexibility required to re-purpose existing software for new and different uses. In order to overcome challenges and develop a lasting infrastructure for future efforts, the basic philosophy in the development of the SU^2 framework has been to ensure:

- An open-source model: while the Aerospace Design Laboratory (ADL) at Stanford University has provided the basic formulation with a reasonable set of initial capabilities, we would like to see contributions from the community to further enhance the capabilities of the suite, and we will ensure that all of these developments are available to all users in the future.
- Portability: SU^2 has been developed using ANSI C++ and only relies on widely-available, well-supported, open-source software including OpenMPI, Metis, and Python. As such, SU^2 is able to run on any computing platform for which a C++ compiler is available.
- Reusability and encapsulation: SU^2 is built so that the main concepts (geometry, grids, solution algorithms, numerical implementations, etc.) are abstracted to a very high-level. This abstraction promotes reusability of the code and enables modifications without incorrectly affecting other portions of the modules in the suite.
- Performance: we have attempted to develop numerical solution algorithms that result in high-performance convergence of the solver in SU^2. Although some level of performance is traded for reusability and encapsulation, the loss in performance is minor.
- Gradient availability: for many applications (optimization, response surface formulations, uncertainty quantification, among others) it is important to obtain gradients of the responses computed by SU^2 to variations of, potentially, very large numbers of design parameters. For this reason, SU^2 relies on adjoint solver implementations that can be used to compute the necessary gradients. In addition, these adjoint solutions can be used to compute functional-driven mesh adaptation techniques.

Using this philosophy within SU^2, we are able to develop both Finite Volume Method (FVM) or FEM solvers, their corresponding adjoint systems, and, if needed, multi-physics solvers that can combine both approaches. The use of a clearly-structured set of classes allows for the easy identification of the main pieces of the code that will need to be re-implemented or enhanced for new models without interfering with the main code. A library of numerical schemes and linear solvers reduces the development time required for a new implementation (e.g., agglomeration multigrid, line-implicit relaxation, and goal-oriented grid adaptation are generic capabilities provided by SU^2 that can be reused in many numerical simulations.) Additionally, the solver structure and parallelization methodology are shared by all the members of the suite. It is important to highlight that the ability to easily integrate these solvers ensures that new features or updated models can be included without affecting the main infrastructure and with a reasonably low degree of difficulty.

At the time of this writing, the SU^2 software suite specializes in high-fidelity PDE analysis and in the design of PDE-constrained systems on unstructured meshes. The suite itself is composed of several C++ analysis modules that handle specific jobs, including:

- SU2_CFD: The main PDE solution module that started primarily as an Euler and RANS CFD solver, but has been modified to treat many other governing equations, including the adjoint equations for many of the supported governing equation systems.
- SU2_DDC: The Domain Decomposition Code, used to prepare SU^2 for computations involving multiple processors.
- SU2_MAC: The Mesh Adaptation Code that can be used to refine the unstructured computational meshes to improve the accuracy of the predictions.
- SU2_GPC: The Gradient Projection Code that allows for the calculation of sensitivities for use in optimization and uncertainty quantification.
- SU2_MDC: The Mesh Deformation Code that can be used to perturb an existing unstructured volume mesh to conform to new surface geometries dictated by either shape optimization processes or aero-structural simulations.
- SU2_PBC: The Periodic Boundary Code, a pre-processor to allow for the solution of PDEs on periodic domains.
- SU2_SMC: The Sliding Mesh Code, a pre-processor that enables the solution of PDEs on meshes that slide (translational or rotational capabilities included) past each other.

Additional modules may be added as further capabilities are needed and included in the software. This structure 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.

SU^2 is under active development in the Aerospace Design Lab (ADL) of the Department of Aeronautics and Astronautics at Stanford University. It has also been released under an open-source license, and it is freely available to the community, so that developers around the world can contribute to and improve the suite. Prior to release, significant efforts were directed at the development of sufficient documentation so that prospective users could get up to speed without interaction with the development team. For this reason a set of tutorials that cover all the basic capabilities of SU^2 was created and is distributed with SU^2. Since the initial release of SU^2 in January of 2012, there have been over $24,000$ visits to the SU^2 web page, more than $3,000$ downloads of the software, many requests for participation in periodic SU^2 workshops, and significant participation in Facebook and Twitter.

The organization of this paper is as follows. Section II describes the object-oriented class structure of SU^2 and the flexibility of the implementation. Details on the numerical modeling are provided in sections III and IV. Section V provides examples of the current functionality and performance of SU^2 using representative simulations and design problems, and finally, the conclusions are summarized in section VI.

It is our intent for this paper to be the main reference for work that uses or enhances the capabilities of SU^2, and for it to serve as a sort of reference manual for researchers and engineers that would like to learn more about the details of the software suite. The current implementation of SU^2 is already very capable, but many additional capabilities are being requested from our development group. We intend to tackle some of them in the near future, but we also look forward to contributions by the entire community in order to make SU^2's future a sustainable one.

II. Code framework

The SU^2 software suite was conceived as a common infrastructure for solving Partial Differential Equation (PDE) problems using the Finite Volume Method (FVM) or Finite Element Method (FEM). The code structure and the high-level time and spatial integration structure is shared by all of the applications, and only specific numerical methods for the convective, viscous and source terms are reimplemented for different models where necessary. There is no fundamental limitation on the number of state variables or the number of governing equation systems that can be solved simultaneously in a coupled or segregated way (other than the physical memory available on a given computer architecture), and the more complicated algorithms and numerical methods (including parallelization, multigrid and linear solvers) have been implemented in such a way that they can be applied without special consideration during the implementation of a new physical model.

The suite is composed of seven C++ software modules that perform a wide range of tasks related to PDE analysis (grid adaptation, grid deformation, surface definition, optimization, etc.). An basic description of each of module is included in Sec. 1 to give a overall perspective of the suite's capabilities. Some modules can be executed individually to perform high fidelity analysis, most notably SU2_CFD, but the real power of the suite lies in the coupling of the modules for performing complex activities including design optimization and adaptive grid refinement. To that end, coupling of the SU^2 software modules can be accomplished using supplied Python scripts that will be also described in Sec. 2. Note also that all of the modules share the same C++ class structure, and thus, for example, all of the grid deformation capabilities can be integrated directly into the CFD solver or used separately as an independent code.

Another key feature of the C++ modules is that each has been designed to separate functionality as much as possible while leveraging 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. Ultimately, this philosophy enables the extension of the suite to a wide variety of PDE analysis and design problems.

A. Software architecture

SU^2 is built to enable vertical integration with optimizers and to reduce the amount of user overhead required for setup. There are five levels of components in the optimization control architecture, and most rely on Python scripts to modify the configuration input, execute lower-level components and post-process any resulting data. To simplify and shorten overhead time during problem setup, all levels start from a common configuration file, which is modified as needed when passed to lower levels. Listed in order from lowest to highest, these levels are:

- Core tools - These contain all of the SU^2 binary executables, which are the core tools of the suite. As input, they take a custom format ASCII configuration file. For output, they write data such as integrated forces, moments and other objectives to an iteration history file, field data to files for plotting, or deformed, adapted, or decomposed meshes in the native format, for instance.

- Solution decomposition/recomposition - Many of the core solvers (i.e., SU2_CFD, SU2_MDC, SU2_GPC, SU2_MAC) can operate in parallel on a partitioned mesh. The management of pre-process mesh decomposition and post-process plot file merging for this data is handled by the 'parallel_computation.py' and 'parallel_deformation.py' Python scripts ('parallel_adaptation.py' is under development).

- Sensitivity analysis - This level manages the pre- and post-processing needed for calculating performance sensitivities with respect to user-specified design variables. Both continuous adjoint and finite differencing approaches have been implemented, and the discrete and hybrid adjoint approaches are under development. For the adjoint approach, both a direct and adjoint solution are computed, and the resulting adjoint surface sensitivities must be projected into the design space during a post-processing step. In the case of finite differencing, multiple, but independent, evaluations of the direct problem are required before the performance sensitivities can be calculated.

- Design evaluation - For easier integration with optimization packages, SU^2 has a design management class that wraps a black box around the previous components and only takes design vectors for input.

This interprets special configuration file options for design variables which allow it to set up mesh deformation. When it receives a design vector from the optimizer, it then executes mesh deformation, direct solution, and sensitivity analyses as needed, and then finally returns performance data. As it operates, it archives restart and plot data in an organized folder structure, which may be useful for secondary analyses or debugging. Evaluations of multiple design requests can be submitted in parallel if the resources are available, for example on a high-performance computing cluster.

- Design optimization - Single-objective design optimization is the highest level of architecture that we have developed at the moment. Two optimization strategies have been adopted for use with SU^2. The first is gradient-based optimization using SciPy's SLSQP optimizer, which adds complexity by requiring separate function handles for the objective function, constraints and their sensitivities. The second is surrogate based optimization, where an in-house gradient-enhanced Gaussian Process Regression based optimizer is used.

1. C++ software modules

The core tools of the SU^2 suite are the C++ modules, a brief description of each binary is presented below:

- SU2_CFD (Computational Fluid Dynamics Code) - Solves direct, adjoint (steady or unsteady) problems for the Euler, Navier-Stokes and Reynolds-Averaged Navier-Stokes (RANS), plasma, free-surface, electrostatic, etc., equation sets. SU2_CFD can be run serially or in parallel using MPI. It uses either FVM or FEM 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, agglomeration multigrid, linelet and low speed preconditioning, and Krylov space methods for solving linear systems. The capabilities of this tool are the subject of much of this article.

- 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 sensitivities computed using SU2_CFD, the flow solution and the definition of the geometrical design variables to evaluate the derivative of a particular functional (e.g. drag, lift, etc.).

- SU2_MDC (Mesh Deformation Code) - Computes the geometrical deformation of surfaces within the computational mesh and the surrounding volumetric grid. Once the type of deformation is defined, SU2_MDC performs the grid deformation using different strategies. Three-dimensional deformations use a method called Free Form Deformation (FFD), while two-dimensional problems typically use bump functions, such as Hicks-Henne.

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

- SU2_DDC (Domain Decomposition Code) - Partitions the specified volumetric grid for use with several of the other core tools when performing simulation or design in parallel. SU2_DDC is built around the METIS[48] software that will identify and assign nodes to each processor for achieving load balancing with minimal communication (edge cuts). Once that information is received, SU2_DDC prepares the communication between nodes on different partitions and generates the appropriate computational grid partitions required by the other core tools for executing in parallel.

- SU2_PBC (Periodic Boundary Code) - Creates ghost cells in the computational domain for performing simulations with periodic boundary conditions and outputs a new mesh containing the proper communication structure between periodic faces. This module must be run prior to SU2_CFD for any simulation the uses such boundary conditions.

- SU2_SMC (Sliding Mesh Code) - Creates ghost cells in the computational domain for performing simulations with sliding surfaces and outputs a new multi-zone mesh containing the proper communication structure between sliding interfaces. As in the case of SU2_PBC, this module must be run prior to SU2_CFD.

The C++ modules share a common class structure to increase the flexibility of the overall SU^2 suite. A description of the main classes of the code is presented in Sec. B.

2. *Python wrapping*

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 most used scripts is provided below:

- High fidelity analysis scripts. These scripts have been designed to enhance the flexibility of the SU^2 framework. More specifically, they simplify the execution of parallel tasks, grid adaptation, or the interface with other software.
  - parallel_computation.py - Handles the setup and execution of parallel CFD jobs on multi-core or cluster computing architectures. The script calls SU2_DDC to partition the grid for the specified number of processors then executes SU2_CFD in parallel.
  - parallel_deformation.py - Handles the setup and execution of parallel mesh deformation jobs on multi-core or cluster computing architectures. The script calls SU2_DDC to partition the grid for the specified number of processors then executes SU2_MDC in parallel.
  - 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.
  - libSU2.py - Contains definitions for commonly used configuration and data file i/o.
  - libSU2_mesh.py - Contains definitions for accessing SU^2 native mesh file data.
  - tasks_general.py - Contains a class structure for abstracting the execution of major SU^2 tasks.
  - tasks_su2.py - Contains project and job class definitions for the non-sequential execution of SU^2 tasks.
  - task_project.py - Performs top-level execution of SU^2 projects which can be wrapped by various optimizers.
  - patient_designspace.py - For use on a computing cluster, it submits and collects data from multiple SU^2 jobs.
- Optimal shape design scripts. These scripts have been designed to automate the optimal shape design process which includes functional and gradient computation, mesh deformation, and an optimization algorithm.
  - continuous_adjoint.py - Automatically computes the sensitivities of a specified functional with respect to design parameter perturbations (specified in the SU2_CFD configuration file) using the adjoint method. The SU2_CFD and SU2_GPC modules are called to perform the analysis.
  - finite_differences.py - Automatically computes the sensitivities of a specified functional with respect to design parameter perturbations using a finite difference method. As with the continuous_adjoint.py script, design variable information is read from the configuration file and SU2_CFD is called repeatedly to calculate the appropriate gradient elements.
  - shape_optimization.py - Orchestrates all SU^2 modules to perform shape optimization. The choice of objective function, design variables and additional module settings specifying the optimization problem are controlled through options in the configuration file.
- Automatic differentiation scripts. A combination of python scripts and the automatic differentiation tool Tapenade[26] can be used by developers to create differentiated versions of code sections in SU^2.
  - convert_routines_cpp2c.py - Uses custom comment lines in the SU^2 C++ source code to create a C version of desired routines, replacing the class structure by simply passing required variables to the new routine.
  - differentiate_routines.py - Calls Tapenade to differentiate a C routine.
  - convert_routines_c2cpp.py - Converts a differentiated C routine back to C++ code, restoring the class structure that existed before in the SU^2 source code.
  - insert_routines.py and insert_math.py - Inserts the differentiated C++ code into source files between custom comment lines.
- General utility scripts.

- autotest.py - This script performs automated regression tests of the SU^2 suite after every commit to the central software repository (continuous integration). Established test cases are executed, and the output is compared against known results. Any discrepancies are immediately reported to the developers. These regression tests help ease the integration of new capabilities while preserving previous functionality in the software.
- config_gui.py - This graphical interface is provided to ease the creation and modification of SU^2 configuration files. The script reads the permitted options directly from the source code, groups them based on functionality, and provides a drop-down menu for enumerated types.

B. C++ class structure

The objective of this section is to introduce the C++ class structure of SU^2. The class descriptions below focus on the structure within SU2_CFD (the main component of SU^2), but most of the classes are also used in the other modules. Maximizing the flexibility of the code was a fundamental driver for the design of the class architecture, and an overview of it is shown in Fig. 1.

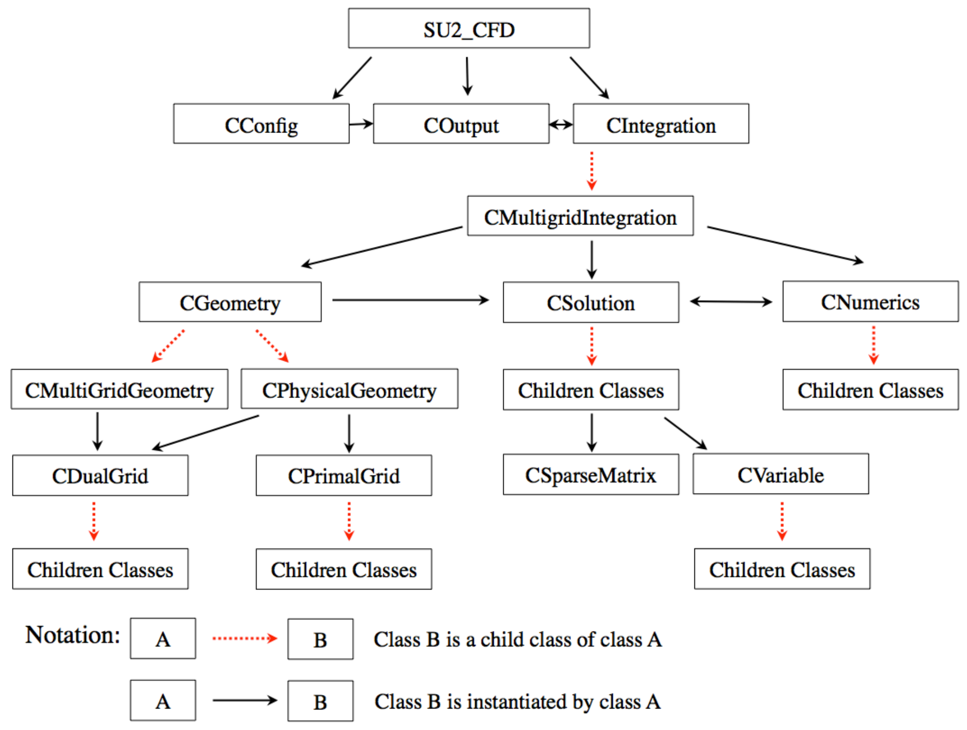


Figure 1. Class hierarchy in SU2_CFD.

As a starting point, the module SU2_CFD instantiates three basic classes, namely:

- CConfig - Reads the problem description including all options and settings from the input file (extension .cfg).
- COutput - Writes the output of the simulation in a user-specified format (Paraview, Tecplot, or comma-separated values).
- CIntegration - Solves the particular governing equations by calling the child classes CMultiGridIntegration, CSingleGridIntegration. This is used for both multigrid or single-grid calculations and connects the subclasses CGeometry, CSolution and CNumerics for performing integration in time and space.

The core capabilities of the computational tool are embedded within the CGeometry, CSolution and CNumerics classes that manage the geometry, the main solver functionality, and the numerical methods, respectively. In the next subsection, these three classes will be discused in detail.

1. *CGeometry class*

This class reads and processes the input mesh file (extension .su2), and it includes several child classes, such as:

- CPhysicalGeometry - Constructs the dual mesh structure from the primal mesh. Note that the FVM formulation in SU^2 is based on this dual mesh.
- CMultiGridGeometry - Creates consecutively coarser meshes from the original input mesh for multigrid calculations.
- CPrimalGrid and CDualGrid - Two classes (see Fig. 2) used for defining all the geometrical characteristics of the primal and dual grids.

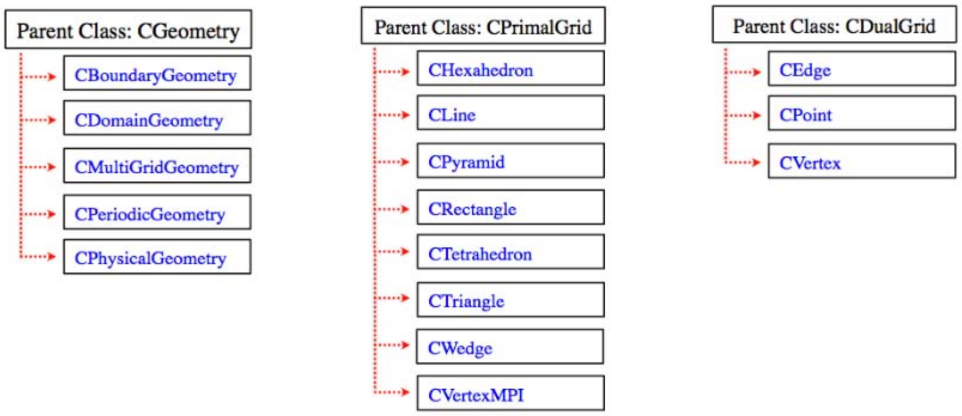


Figure 2. Parent and child classes related to geometry processing.

2. *CSolution class*

In this class, the solution procedure is defined. Each child class of CSolution represents a solver for a particular set of governing equations. One or more of these child classes will be instantiated depending on the desired physics, and several examples are:

- CEulerSolution - For the Euler equations (compressible or incompressible).
- CTurbSolution - For a turbulence model.
- CPlasmaSolution- For the reacting flow equations.
- CAdjEulerSolution - For the adjoint equations of the Euler equations.

These subclasses call several classes in CNumerics to discretize the governing equations and, if necessary, the CSparseMatrix class to solve the linear system of equations and CSysVector to hold and manipulate vectors needed by linear solvers. A detailed list of all child classes found within CSolution is given in Fig. 3, but in general, these subclasses instantiate copies of two other classes:

- CVariable - Used to store variables at every point in the grid, such as the conservative variables. Depending on the system of equations being solved, CVariable instantiates a certain child class and stores a set of variables particular to that problem at each grid node. For example, the CNSVariable child class stores the variables for the Navier-Stokes equations which will include viscosity, while the CEulerVariable child class does not need to store viscosity. A detailed list of all these child classes is given in Fig. 3.
- CSparseMatrix - Stores values for the Jacobians of fluxes and source terms in a sparse matrix structure for implicit calculations. It includes various methods for solving a linear system such as LU-SGS, BiCGSTAB, GMRES, among others, in addition to several preconditioning techniques such as line implicit or Jacobi preconditioning.

- CSysVector - Holds and manipulates vectors needed by linear solvers in combination with CSparseMatrix to store the Jacobian sparse matrix.

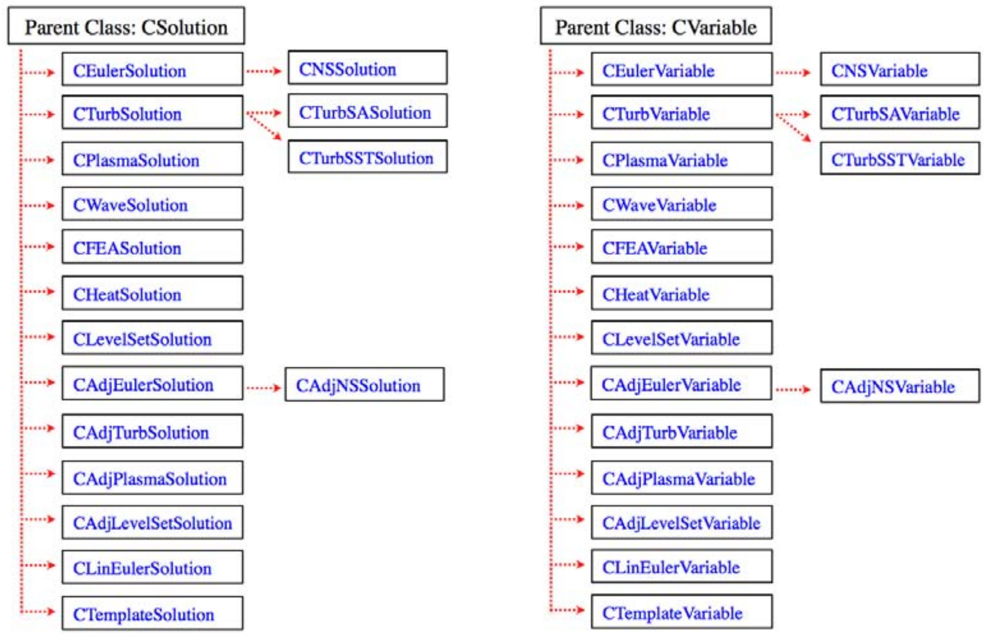


Figure 3. List of child classes for the CSolution and the CVariable classes. A few of the child classes have their own children, such as CEulerSolution, which has the class called CNSSolution that inherits all the characteristics of CEulerSolution in addition to those that are specific to it (viscous methods). The "CTemplate" child class is a template for adding future capabilities.

3. CNumerics class

This class discretizes each system of governing equations using the numerical schemes specified in the input file. There are several child classes which provide a wide range of discretization techniques for convective fluxes, viscous fluxes and any source terms present. For example, if one is interested in solving for a plasma flow, CNumerics would call one child class corresponding to the convective scheme, one corresponding to the viscous terms and a third for the discretization of the chemical source terms.

For instance, in an implicit calculation, methods in these classes calculate the residuals and Jacobians at each node (using the variables stored in the CVariable class) and feed the results back to CSolution which then calls routines within CSparseMatrix to solve the resulting linear system of equations. Fig. 4 shows a list of the various capabilities in the CNumerics class.

As an example of the complete class architecture instantiated for a specific set of governing equations, Fig. 5 shows the structure for a simulation of the RANS equations with explicit time integration. It is important to highlight that these equations are solved as a multiphysics problem (i.e., the mean flow and turbulence model equations are solved separately and coupled). Therefore, two child classes are instantiated from CSolution (CNSSolution and CTurbSolution) as well as from CVariable (CNSVariable and CTurbVariable).

C. File input/output

1. Configuration file

The configuration file is a simple text file (extension .cfg) that sets the options for the SU^2 suite. The configuration file consists of three possible 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

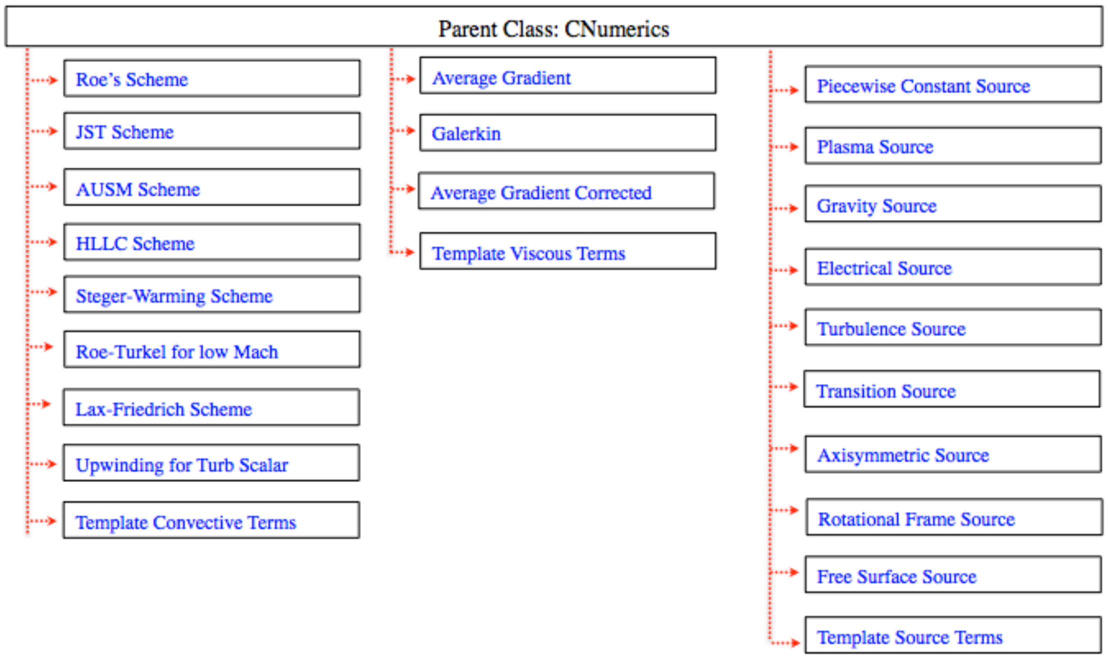


Figure 4. List of child class capabilities found under the CNumerics parent class. The "CTemplate" child class is a template for adding future capabilities.

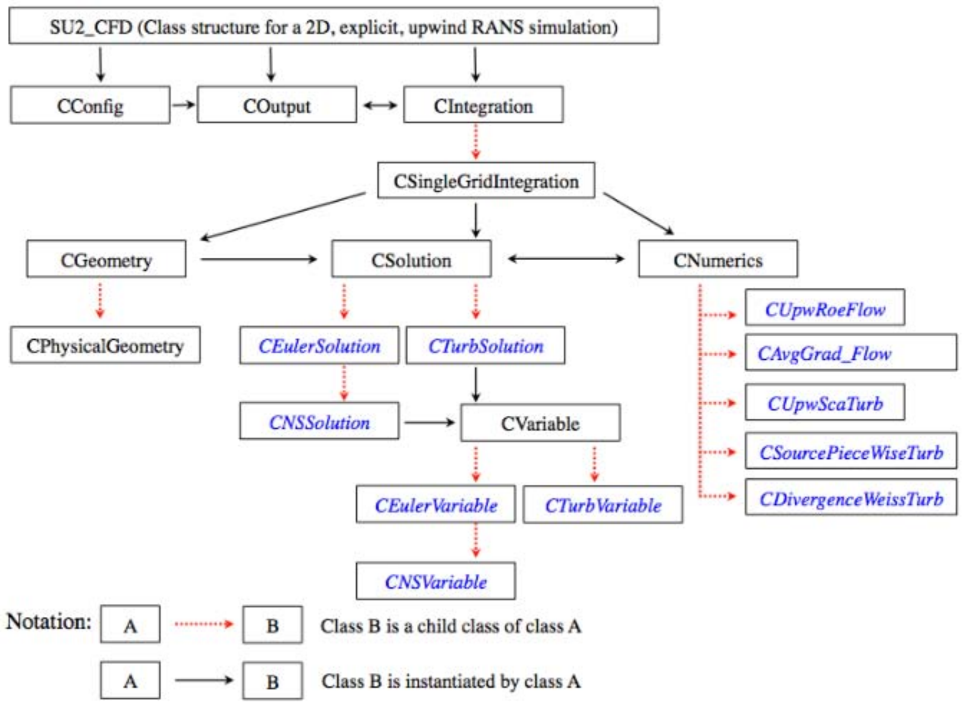


Figure 5. Class hierarchy for solving a two-dimensional RANS problem.

type, a list of multiple data types, or a more complicated structure. The "=" sign must immediately follow the option_name element and is not optional, though spaces and tabs between the various value elements are not significant. Lower, upper, or mixed-case strings are allowed.

- Comments - On a single line, any text appearing after a % is considered a comment and is ignored by SU^2. Additional % signs after the first appearance on any line are not significant.
- 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.

2. *Mesh file native format (.su2)*

In keeping with the open source nature of the project, SU^2 relies mostly on its own native mesh format, which is meant to be simple and readable. SU^2 meshes carry an extension of .su2, and the files are in ASCII format. As an unstructured code, the suite requires information about both the node locations as well as their connectivity, which provides information about the types of elements (triangle, rectangle, tetrahedron, hexahedron, 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 element connectivity is specified in a similar manner as that of the interior elements.

The first line of the .su2 mesh declares the dimensionality of the problem. SU^2 can handle two- or three-dimensional geometries, but note that for two-dimensional simulations, it is recommended that a truly two-dimensional mesh is used (no z-coordinates) rather than a quasi-two-dimensional mesh (one or more cells deep in the third dimension with symmetry boundary conditions).

The next part of the file describes the interior element connectivity. SU^2 is based on unstructured mesh technology and thus supports several element types for both two-dimensional and three-dimensional elements. Unlike for structured meshes where a logical, ordered indexing can be assumed for neighboring nodes and their corresponding cells (rectangles in two-dimensions and hexahedral elements in three-dimensions), for an unstructured mesh, a list of nodes that make up each element must be provided.

On each line in the connectivity section of the mesh file, following an identifier of the element type is a list of the node indices that make up the element. Each triangular element will have three nodes specified, a rectangular element will have four nodes specified, a tetrahedral element will have four nodes specified and so on. The final value is the element index given to each interior element in the mesh.

After the connectivity information for all interior elements, the coordinates for each of the nodes are given. 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 unique index number. The node index numbers are the indices used for specifying the connectivity information for elements. For a two-dimensional mesh, the x and y coordinates are given followed by the index, but a three-dimensional mesh would give x, y, and z, followed by the index.

The final section of the mesh file is a description of all boundaries (which we call markers), including a name (called a tag). For each boundary, the connectivity information is given based on the index given to each node. For a two-dimensional mesh, only line elements are possible along the boundaries, but for a three-dimensional mesh, there are two possible options for boundary elements: triangular and rectangular. First, the number of boundaries, or markers, is specified. Then for each marker, a name, or tag, is specified. This tag can be any string name, and the tag name is used in the configuration file for the solver when specifying boundary conditions. The number of elements on each marker must then be specified before listing the connectivity information as was done for the interior mesh elements at the start of the file. Again, the unique VTK[30] identifier is given at the start of each line followed by the node list for that element.

3. *Solution and restart files*

SU^2 is currently capable of outputting solution files that can be visualized with either ParaView (.vtk) or Tecplot (.plt). At the end of each simulation, the software creates several files that contain all of the necessary information for both visualization and restart, and the default names of these files can be changed in the configuration file. For unsteady flows, activating the unsteady writing option in the configuration file will automatically increment the file names with the corresponding time step number and prepare the file for animation within visualization software.

- For a direct flow solution, these default files names are:
  - flow.plt or flow.vtk - Full volume flow solution.
  - surface_flow.plt or surface_flow.vtk - Flow solution on specified surfaces.
  - surface_flow.csv - Comma separated values (.csv) file containing values on specified surfaces.
  - restart_flow.dat - Restart file in a native format for restarting simulations in SU^2.
  - history.plt or history.csv - File containing the convergence history information.
- For adjoint solutions, the default file names are slightly different:
  - adjoint.plt or adjoint.vtk - Full volume adjoint solution.
  - surface_adjoint.plt or surface_adjoint.vtk - Adjoint solution on specified surfaces.
  - surface_adjoint.csv - Comma separated values (.csv) file containing values on specified surfaces.
  - restart_adj_cd.dat - Restart file in a native format for restarting simulations in SU^2. Note that the name of the objective function appears in the file name.
  - history.plt or history.csv - File containing the convergence history information.

It is important to highlight that SU^2 also uses a simple and readable format for the restart files. The SU^2 solution file carries the extension .dat, and the files are in ASCII format. The solution, and eventually the residual, is provided at each node in the numerical grid, and no information about the connectivity or coordinates is included in the file. For the sake of clarity, the node index is provided at the beginning of each line, though this index value is not used by the code and the ordering of the points is (and must be kept) the same as in the mesh file. The restart files are used not only to restart the code from a previous solution but also to run adjoint simulations, which require a converged flow solution.

III. Modeling

The flexible class structure of SU^2 has been designed to solve problems defined on a domain $\Omega \subset \mathbb{R}^3$, delimited by disconnected boundaries. In particular, the PDE system resulting from physical modeling of the problem should have the following structure:

$$\partial_t U + \nabla \cdot \vec{F}^c - \nabla \cdot \vec{F}^v = Q \quad \text{in } \Omega,\, t > 0 \tag{1}$$

with appropriate boundary and temporal conditions that will be problem-dependent. In this general framework, U represents the vector of state variables, $\vec{F}^c(U)$ are the convective fluxes, $\vec{F}^v(U)$ are the viscous fluxes and $Q(U)$ is a generic source term. In this section, some of the most important physical systems already implemented in SU^2 will be described using Eq. 1 as a baseline PDE.

A. Reynolds-averaged Navier-Stokes (RANS) equations

Several forms of the RANS equations have been implemented in SU^2 (compressible, incompressible, Arbitrary Lagrangian-Eulerian, etc.), and a brief description of the physics involved and the corresponding governing equations is given below for each. Naturally, both the laminar Navier-Stokes and Euler equations are also available in the code as subsets of the RANS equations by disabling turbulence modeling and by completely removing viscosity, respectively.

1. Compressible formulation

Classic aeronautical applications assume that the air is governed by the compressible Navier-Stokes equations[50] which describe the conservation of mass, momentum, and energy in a viscous fluid. In the SU^2 framework, the vector of conservative variables is $U = (\rho, \rho v_1, \rho v_2, \rho v_3, \rho E)^{\mathsf{T}}$, where ρ is the density, E is the total energy per unit mass, and $\vec{v} = (v_1, v_2, v_3) \in \mathbb{R}^3$ is the flow velocity in a Cartesian coordinate system. In this particular model, convective and viscous fluxes are then given by

$$\vec{F}^c_i = \begin{pmatrix} \rho v_i \\ \rho v_i v_1 + P\delta_{i1} \\ \rho v_i v_2 + P\delta_{i2} \\ \rho v_i v_3 + P\delta_{i3} \\ \rho v_i H \end{pmatrix}, \quad \vec{F}^v_i = \begin{pmatrix} \cdot \\ \tau_{i1} \\ \tau_{i2} \\ \tau_{i3} \\ v_j \tau_{ij} + \mu^*_{tot} C_p \partial_i T \end{pmatrix}, \quad i = 1, \ldots, 3 \tag{2}$$

where P is the static pressure, H is the fluid enthalpy, δ_{ij} is the Kronecker delta function, and the viscous stresses may be written as $\tau_{ij} = \mu_{tot}\left(\partial_j v_i + \partial_i v_j - \frac{2}{3}\delta_{ij} \nabla \cdot \vec{v}\right)$. Recall that latin indices i, j denote 3-D Cartesian coordinates with repeated indices implying summation.

In these formulas, C_p is the specific heat at constant pressure, $T = P/(R\rho)$ is the temperature and R is the gas constant, such that for an ideal gas, $C_p = \gamma R/(\gamma - 1)$, with γ being a constant. In order to close the system of equations, the dynamic viscosity, μ_{dyn}, is assumed to satisfy Sutherland's law,[97] the turbulent viscosity μ_{tur} is computed via a turbulence model, and

$$\mu_{tot} = \mu_{dyn} + \mu_{tur}, \quad \mu^*_{tot} = \frac{\mu_{dyn}}{Pr_d} + \frac{\mu_{tur}}{Pr_t}, \tag{3}$$

where Pr_d and Pr_t are the dynamic and turbulent Prandtl numbers, respectively.

The compressible RANS solver in SU^2 currently supports the following boundary condition types: Euler (flow tangency) and symmetry wall, no-slip wall (adiabatic and isothermal), far-field and near-field boundaries, characteristic-based inlet boundaries (stagnation, mass flow, or supersonic conditions prescribed), characteristic-based outlet boundaries (back pressure prescribed), periodic boundaries, nacelle inflow boundaries (fan face Mach number prescribed), and nacelle exhaust boundaries (total nozzle temp and total nozzle pressure prescribed).

An important consideration when solving the Navier-Stokes equations is the non-dimensionalization. The particular scheme chosen for SU^2 can be found in Tables 1, 2 and 3, and is based on the internal document "Non-dimensionalization of the Navier-Stokes Equations" written by Prof. Feng Liu.

Variables	SU^2 value	SI
Length	l_{ref} (input)	m
Pressure	p_{ref} (input)	N/m^2
Density	ρ_{ref} (input)	kg/m^3
Temperature	T_{ref} (input)	K
Velocity	$u_{ref} = \sqrt{p_{ref}/\rho_{ref}}$	m/s
Time	$t_{ref} = l_{ref}/u_{ref}$	s
Dynamic viscosity	$\mu_{ref} = \rho_{ref}\, u_{ref}\, l_{ref}$	kg/(m.s)
Rotor speed	$\Omega_{ref} = u_{ref}/l_{ref}$	1/s
External body force	$b_{ref} = u^2_{ref}/l_{ref}$	m/s^2

Table 1. Basic independent variables for which the reference values can be arbitrarily chosen.

Variables	SU^2 value	SI
Kinematic viscosity	$\nu_{ref} = \mu_{ref}/\rho_{ref}$	m^2/s
Strain rate	$S_{ref} = u_{ref}/l_{ref}$	1/s
Stress	$\tau_{ref} = p_{ref}$	N/m2
Specific energy	$e_{ref} = u^2_{ref}$	J/kg
Specific enthalpy	$h_{ref} = e_{ref}$	J/kg
Specific entropy	$s_{ref} = e_{ref}/T_{ref}$	J/(kg.K)
Heat flux	$q_{ref} = \rho_{ref}\, e_{ref}\, u_{ref}$	J/(m^2.s)
Gas constant	$R_{ref} = e_{ref}/T_{ref}$	J/(kg.K)
Heat capacity (constant pressure)	$c_{p_{ref}} = R_{ref}$	J/(kg.K)
Heat capacity (constant volume)	$c_{v_{ref}} = R_{ref}$	J/(kg.K)
Heat conductivity	$k_{ref} = c_{p_{ref}}\, \mu_{ref}$	W/(K.m)
Turbulent kinetic energy	$k_{ref} = u^2_{ref}$	J/kg
Turbulent specific dissipation	$\omega_{ref} = u_{ref}/l_{ref}$	1/s

Table 2. Derived variables whose reference values are determined from the choice of the basic independent variables in Table 1.

2. Incompressible formulation

The incompressible solver in SU^2 is based on the artificial compressibility formulation developed by Chorin[13] (valid for steady-state only), and it uses the following set of state variables $U = (P, \rho v_1, \rho v_2, \rho v_3)^\mathsf{T}$, where P is the pressure and $\vec{v} = (v_1, v_2, v_3) \in \mathbb{R}^3$ is the flow velocity in a Cartesian coordinate system. The convective and viscous fluxes are then given by

$$\vec{F}^c_i = \begin{pmatrix} \beta^2 v_i \\ \rho v_i v_1 + P\delta_{i1} \\ \rho v_i v_2 + P\delta_{i2} \\ \rho v_i v_3 + P\delta_{i3} \end{pmatrix}, \quad \vec{F}^v_i = \mu_{tot} \begin{pmatrix} \cdot \\ \partial_i v_1 \\ \partial_i v_2 \\ \partial_i v_3 \end{pmatrix}, \quad Q = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ -\frac{\rho}{Fr^2} \end{pmatrix}, \quad i = 1, \dots, 3 \tag{4}$$

where the artificial compressibility parameter is denoted by β^2, Fr is the Froude number, and the total viscosity is computed as $\mu_{tot} = \mu_{dyn} + \mu_{tur}$. The density and viscosity are inside the differential operators because their value will change across a hypothetical free-surface interface (already implemented in SU^2).

The incompressible solver in SU^2 currently supports the following boundary condition types: Euler (flow tangency) wall, no-slip wall (adiabatic), far-field and symmetry boundaries, inlet boundaries (total conditions or mass flow prescribed), and outlet boundaries (back pressure prescribed).

Variables	SU^2 value
Ratio of specific heats	$\gamma = C_p/C_v$ (input)
Dimensionless gas constant	$R = R^*/R_{ref}$ (input R^*)
Molecular Prandtl number	$Pr_l = C_p\mu/k$ (input)
Turbulent Prandtl number	$Pr_t = C_p\mu_t/k_t$ (input)
Strouhal number	$(St)_{ref} = l_{ref}/(u_{ref}\, t_{ref}) = 1$
Euler number	$(Eu)_{ref} = p_{ref}/(\rho_{ref}\, u_{ref}^2) = 1$
Reynolds number	$(Re)_{ref} = \rho_{ref}\, u_{ref}\, l_{ref}/\mu_{ref} = 1$
Rossby number	$(Ro)_{ref} = u_{ref}/(\Omega_{ref}\, l_{ref}) = 1$
Froude number	$(Fr)_{ref} = u_{ref}/\sqrt{b_{ref}\, l_{ref}} = 1$

Table 3. Non-dimensional parameters and coefficients based on the reference variables.

3. *Turbulence modeling*

In accord with the standard approach to turbulence modeling based upon the Boussinesq hypothesis,[98] which states that the effect of turbulence can be represented as an increased viscosity, the total viscosity is divided into a laminar, μ_{dyn}, and a turbulent, μ_{tur}, component. The laminar, or dynamic, viscosity is usually taken to be a function of temperature only, $\mu_{dyn} = \mu_{dyn}(T)$, whereas μ_{tur} is obtained from a suitable turbulence model involving the flow state, and a set of new variables.

Spalart-Allmaras (S-A) Model: In the case of the one-equation Spalart-Allmaras[84] turbulence model, the turbulent viscosity is computed as

$$\mu_{tur} = \rho\hat{\nu} f_{v1}, \quad f_{v1} = \frac{\chi^3}{\chi^3 + c_{v1}^3}, \quad \chi = \frac{\hat{\nu}}{\nu}, \quad \nu = \frac{\mu_{dyn}}{\rho}. \tag{5}$$

The new variable $\hat{\nu}$ is obtained by solving a transport equation where the convective, viscous, and source terms are given as follows:

$$\vec{F}^c = \vec{v}\hat{\nu}, \quad \vec{F}^v = -\frac{\nu+\hat{\nu}}{\sigma}\nabla\hat{\nu}, \quad Q = c_{b1}\hat{S}\hat{\nu} - c_{w1} f_w \left(\frac{\hat{\nu}}{d_S}\right)^2 + \frac{c_{b2}}{\sigma}|\nabla\hat{\nu}|^2, \tag{6}$$

where the production term $\hat{S}$ is defined as $\hat{S} = |\vec{\omega}| + \frac{\hat{\nu}}{\kappa^2 d_S^2} f_{v2}$, $\vec{\omega} = \nabla \times \vec{v}$ is the fluid vorticity, d_S is the distance to the nearest wall, and $f_{v2} = 1 - \frac{\chi}{1+\chi f_{v1}}$. The function f_w is computed as $f_w = g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{1/6}$, where $g = r + c_{w2}(r^6 - r)$ and $r = \frac{\hat{\nu}}{\hat{S}\kappa^2 d_S^2}$. Finally, the set of closure constants for the model is given by

$$\sigma = 2/3,\ c_{b1} = 0.1355,\ c_{b2} = 0.622,\ \kappa = 0.41,\ c_{w1} = \frac{c_{b1}}{\kappa^2} + \frac{1+c_{b2}}{\sigma},\ c_{w2} = 0.3,\ c_{w3} = 2,\ c_{v1} = 7.1. \tag{7}$$

The physical meaning of the far-field boundary condition for the turbulent viscosity is the imposition of some fraction of the laminar viscosity at the far-field. On viscous walls, $\hat{\nu}$ is set to zero, corresponding to the absence of turbulent eddies very near to the wall.

Menter Shear Stress Transport (SST) Model: The Menter SST turbulence model[62] is a two equation model for the turbulent kinetic energy k and specific dissipation ω that consists of the blending of the traditional $k-\omega$ and $k-\epsilon$ models. The definition of the eddy viscosity, which includes the shear stress limiter, is shown below.

$$\mu_{tur} = \frac{\rho a_1 k}{max(a_1\omega, SF_2)}. \tag{8}$$

where $S = \sqrt{2S_{ij}S_{ij}}$ and F_2 is the second blending function.

The convective, viscous and source terms for the turbulent kinetic energy follow,

$$\vec{F}^c = \rho k\vec{v}, \quad \vec{F}^v = -(\mu_{dyn} + \sigma_k\mu_{tur})\nabla k, \quad Q = P - \beta^*\rho\omega k, \tag{9}$$

where P is the production of turbulent kinetic energy. The convective, viscous and source terms for the specific dissipation follow,

$$\vec{F}^c = \rho\omega\vec{v}, \quad \vec{F}^v = -(\mu_{dyn} + \sigma_\omega\mu_{tur})\nabla\omega, \quad Q = \frac{\gamma}{\nu_t}P - \beta^*\rho\omega^2 + 2*(1-F_1)\frac{\rho\sigma_{\omega 2}}{\omega}\nabla k\nabla\omega, \tag{10}$$

where F_1 is the first blending function.

The values for the constants and the forms for the blending functions and auxiliary relations are detailed in Rumsey[76] and Menter.[62] The Menter SST and the S-A turbulence models are the two most common and widely used RANS models for the analysis and design of engineering applications affected by turbulent flows.

4. *Transition model*

The $\gamma - \overline{Re_{\theta t}} - SA$ transition model[61] is an adaptation of the model of Langtry and Menter.[51] It is a correlation-based model that augments the Spalart-Allmaras turbulence model with two equations. The first is for intermittency, γ, a quantity that varies from 0 in laminar regions to 1 in fully turbulent regions of the flow. The convective, viscous, and source terms for the transport of γ are

$$\vec{F}^c = \rho\gamma\vec{v}, \quad \vec{F}^v = -\left(\mu_{dyn} + \frac{\mu_{tur}}{\sigma_f}\right)\nabla\gamma, \quad Q = P_\gamma - D_\gamma, \tag{11}$$

where the source terms P_γ and D_γ are obtained from correlations. The second equation is for local onset momentum thickness Reynolds number, $\overline{Re_{\theta t}}$. Its convective, viscous, and source terms are

$$\vec{F}^c = \rho\overline{Re_{\theta t}}\vec{v}, \quad \vec{F}^v = -\sigma_{\theta t}(\mu_{dyn} + \mu_{tur})\nabla\overline{Re_{\theta t}}, \quad Q = P_{\theta t}, \tag{12}$$

where once again the production term $P_{\theta t}$ is obtained from correlation.

The source terms for the Spalart-Allmaras equation are modified to depend on intermittency as follows

$$Q = \gamma_{eff}c_{b1}\hat{S}\hat{\nu} - \max(\min(\gamma,\beta),1.0)\left[c_{w1}f_w\left(\frac{\hat{\nu}}{d_S}\right)^2\right] + \frac{c_{b2}}{\sigma}|\nabla\hat{\nu}|^2. \tag{13}$$

For details, including correlations and closure constants, the reader is referred to Medida et al.[61] and Langtry et al.[51]

5. *Rotating frame and Arbitrary Lagrangian-Eulerian (ALE) formulations*

When simulating fluid flow about certain aerodynamic bodies that operate under an imposed steady rotation, including many turbomachinery, propeller, and rotor applications, it can be advantageous to transform the system of flow equations into a reference frame that rotates with the body of interest.[19, 20, 29] With this transformation, a flow field that is unsteady when viewed from the inertial frame can be solved for in a steady manner, and thus more efficiently, without the need for grid motion.

After performing the appropriate transformation into a reference frame that rotates with a steady angular velocity, $\vec{\omega} = (\omega_1, \omega_2, \omega_3) \in \mathbb{R}^3$, and a specified rotation center, $\vec{r}_o = (x_o, y_o, z_o) \in \mathbb{R}^3$, the convective fluxes of the compressible equations are modified and a new source terms appears as

$$\vec{F}^c = \begin{pmatrix} \rho(\vec{v} - \vec{u}_r) \\ \rho\vec{v}\otimes(\vec{v} - \vec{u}_r) + \bar{\bar{I}}P \\ \rho H(\vec{v} - \vec{u}_r) + P\vec{u}_r \end{pmatrix}, \quad Q = \begin{pmatrix} 0 \\ -\rho(\vec{\omega}\times\vec{v}) \\ 0 \end{pmatrix},$$

where ρ is the fluid density, $\vec{v} = (v_1, v_2, v_3) \in \mathbb{R}^3$ is the absolute flow velocity, H is the total enthalpy per unit mass, P is the static pressure, and $\vec{u}_r$ is the velocity due to rotation ($\vec{u}_r = \vec{\omega}\times\vec{r}$). Here, $\vec{r}$ is the position vector pointing from the rotation center to a point (x, y, z) in the flow domain, or $\vec{r} = ((x - x_o), (y - y_o), (z - z_o))$. The velocity due to rotation is also sometimes called the whirl velocity. The viscous terms remain the same as in the inertial frame. The modified flow tangency wall boundary condition is $(\vec{v} - \vec{u}_r)\cdot\vec{n}_S = 0$ for inviscid flows, where $\vec{n}_S$ is the unitary normal vector to the surface S, and the no-slip wall boundary condition is $\vec{v} = \vec{u}_r$.

It is important to note that not all simulations of rotating bodies can benefit from this solution approach. The flow field must be steady in the rotating frame, and some conditions or geometric features, such as relative surface motion, can cause unsteadiness for rotating bodies. Furthermore, many unsteady flows of interest, such as pitching, plunging, or rotating surfaces, require solutions on arbitrarily moving grids. For that reason, an Arbitrary Lagrangian-Eulerian (ALE) formulation[21,31] has also been implemented in SU^2. In this formulation, the convective fluxes are adjusted to take into account the grid motion:

$$\vec{F}^c = \begin{pmatrix} \rho(\vec{v} - \vec{u}_x) \\ \rho\vec{v} \otimes (\vec{v} - \vec{u}_x) + \bar{\bar{I}}P \\ \rho E(\vec{v} - \vec{u}_x) + P\vec{v} \end{pmatrix}, \tag{14}$$

where $\vec{u}_x$ is the local velocity for a domain in motion. The viscous terms again remain the same as in the fixed domain, but unlike the rotating frame formulation, there is no additional source term. The modified flow tangency wall boundary condition is $(\vec{v} - \vec{u}_x) \cdot \vec{n}_S = 0$ for inviscid flows, and the no-slip wall boundary condition is $\vec{v} = \vec{u}_x$.

B. Free-surface solver

The free-surface solver in SU^2 is based on the incompressible solver while allowing the density and the viscosity to change depending on the location of the free-surface. To identify the free-surface, a level set[81] function, ϕ, is used to track the interface between the gas and the liquid.[68,85] In particular, the interface will be the zero level set of ϕ, and the level set function will be positive in the gas and negative in the liquid. The level set variable, ϕ, should satisfy the following transport equation (written in terms of our baseline model):

$$\vec{F}^c(\vec{v}, \phi) = \phi\,\vec{v}, \quad Q = (\phi - \phi_0)\,\xi, \tag{15}$$

where $\phi_0 = \phi_0(\vec{x})$ is the initial distance from the free-surface to the boundaries and ξ is a damping factor only activated near the inlet and outlet. The level set equation simply states that the interface moves with the fluid. The values of the density and laminar viscosity are defined using an approximation of the Heaviside function $H = H(\phi, \epsilon)$:

$$H(\phi) = \begin{cases} 1 & \text{if } \phi < -\epsilon, \\ 1 - \frac{1}{2}\left[1 + \frac{\phi}{\epsilon} + \frac{1}{\pi}\sin(\pi\phi/\epsilon)\right] & \text{if } |\phi| \le \epsilon, \\ 0 & \text{if } \phi > \epsilon, \end{cases} \tag{16}$$

where ϵ is a measure of the interface thickness. Finally, density and viscosity are computed as:

$$\rho(\phi) = H(\phi) + \left(\frac{\rho_g}{\rho_l}\right)(1 - H(\phi)), \quad \mu(\phi) = H(\phi) + \left(\frac{\mu_g}{\mu_l}\right)(1 - H(\phi)), \tag{17}$$

where the subscripts g and l denote gas and liquid, respectively.

C. Wave equation

1. *Ffowcs Williams-Hawkings (FW-H) equation*

For aeroacoustic problems, perturbations in density, ρ' where $\rho'(\vec{x}, t) = \rho(\vec{x}, t) - \rho_\infty$ being ρ_∞ the flow density at the infinity, form the longitudinal waves that are perceived as sound. Consider an aerodynamic body immersed in an unbounded volume of fluid, Ω. A fictitious, near-field control surface, Γ_{nf}, is placed near the body, and the fluid domain is therefore divided into two regions, labeled Ω_1 and Ω_2. As a mathematical convenience, we define the shape of Γ_{nf} by a function, $f = 0$, such that $f < 0$ inside the body and $f > 0$ outside the body. We also assume that $\nabla(f)$ is in the direction of the outward normal, such that $\nabla(f) = \vec{n}_{\Gamma_{nf}}|\nabla(f)|$. Furthermore, Γ_{nf} can be in motion with arbitrary boundary velocity, $\vec{u}_b$.

Following the derivation by Ffowcs Williams and Hawkings,[23] the permeable surface version of the FW-H equation can be formulated as

$$\begin{aligned} \frac{\partial^2 \rho'}{\partial t^2} - c^2\nabla^2\rho' &= \frac{\partial}{\partial t}\left\{[\rho(\vec{v} - \vec{u}_b) + \rho_\infty\vec{u}_b] \cdot \nabla(f)\delta(f)\right\} \\ &- \nabla \cdot \left\{\left[\rho\vec{v} \otimes (\vec{v} - \vec{u}_b) + \bar{\bar{I}}p'\right] \cdot \nabla(f)\delta(f)\right\} + \nabla^2\mathbb{T} \;\text{ in } \Omega,\; t > 0, \end{aligned} \tag{18}$$

where, in this case, $\delta(f)$ is the Dirac delta function involving the near-field surface. The nomenclature $[\,]_{(1)}^{(2)}$ represents the jump between regions Ω_2 and Ω_1. The terms appearing on the right hand sides can be thought of as sources concentrated at the surface, Γ_{nf}, which are required to maintain conservation for the unbounded fluid. The speed of sound c is taken as a constant, and $\mathbb{T} = \rho\vec{v}\otimes\vec{v} + P - \bar{\bar{I}}a^2\rho'$ is the Lighthill stress tensor that represents the difference between the stress state in the real fluid and that in the acoustic medium.

Here we see that the propagation of sound generated by aerodynamic surfaces in arbitrary motion is governed by the wave equation, and the sound generation processes are composed of three types of sources on the right hand side: a mass displacement effect by the surface with monopole character (thickness noise, first term), a surface distribution of dipoles (loading noise, second term), and a distribution of quadrupole noise sources throughout the volume exterior to the surface (third term). For simplicity, the source terms will be lumped together as a single term, $\mathcal{Q}$, giving the following expression for the convective, viscous, and source terms:

$$U = \begin{pmatrix} \rho' \\ \xi \end{pmatrix}, \quad \vec{F}_i^v = \begin{pmatrix} \cdot \\ c^2\partial_i\rho' \end{pmatrix}, \quad Q = \begin{pmatrix} \xi \\ \mathcal{Q} \end{pmatrix}, \quad i = 1,\dots,3 \tag{19}$$

where the time discretization is performed by splitting the original equation into two partial differential equations and only a first order time derivative appears.

2. Linear elasticity equation

The equations of linear elasticity[100] govern small displacements, $V = (u_1, u_2, u_3)^T$, of an elastic solid subject to body forces and surface tractions. If the inertial terms are included, the linear elasticity equation can be written as

$$\frac{\partial^2 V}{\partial t^2} - \nabla\sigma = f \quad \text{in } \Omega,\ t > 0, \tag{20}$$

with f being a body force and σ the stress tensor given in terms of the strain tensor, ϵ, by the constitutive relation

$$\sigma = \lambda Tr(\epsilon)I + 2\mu\epsilon, \quad \epsilon = \frac{1}{2}(\nabla u + \nabla u^T), \quad \lambda = \frac{\nu E}{(1+\nu)(1-2\nu)}, \quad \mu = \frac{E}{2(1+\nu)}, \tag{21}$$

where Tr is the trace, λ and μ are the Lamé constants, E is the Young's modulus, and ν is Poisson's ratio. $E > 0$ may be thought of as the stiffness of the material, where a large value of E indicates rigidity. Poisson's ratio, ν, is a measure of how much the material shrinks in the lateral direction as it extends in the axial direction. The following expression for the convective, viscous, and source terms will be used:

$$U = \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}, \quad \vec{F}_i^v = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \\ \sigma_{i1} \\ \sigma_{i2} \\ \sigma_{i3} \end{pmatrix}, \quad Q = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ f_1 \\ f_2 \\ f_3 \end{pmatrix}. \quad i = 1,\dots,3 \tag{22}$$

where the time discretization is performed by splitting the original equation into two partial differential equations and only a first order time derivative appears.

D. Heat equation

The heat equation describes the distribution of heat in a given region over time. Using the baseline equation, the different terms of the heat equation are

$$U = T, \quad \vec{F}_i^v = \alpha\partial_i T, \quad Q = q, \quad i = 1,\dots,3 \tag{23}$$

where T is the temperature, q is a heat source, and $\alpha = k/C_p\rho$ is the thermal diffusivity, a material-specific quantity depending on the thermal conductivity k, the mass density ρ, and the specific heat capacity C_p. Dirichlet or Neumann boundary conditions are admissible in this model.

E. Gauss's law

Gauss's law is a part of the Maxwell's equations and relates the distribution of electric charge to the resulting electric field. Using the baseline equation, the different terms of the Gauss's law equation are

$$\vec{F}^v = -\vec{E}, \quad Q = \frac{\rho}{\epsilon_0}, \tag{24}$$

where $\vec{E}$ is the electric field, ρ is the net electric charge density, and ϵ_0 is the permittivity of free space. Dirichlet or Neumann boundary conditions are admissible in this model.

F. Plasma equations

Due to the complexity of the plasma model chosen for use in SU^2, in this section we will present only a basic description, and the interested reader is referred to several publications using the present formulation.[3, 10, 16, 52, 54, 57, 70, 92] SU^2 solves a full set of flow equations for each chemical species in the plasma and does not use any diffusion models to calculate species velocity. The flow equations are tightly coupled with Gauss's law for calculation of the electric field generated by any local separation of charge. Given a plasma with nS number of chemical species, the different terms are given as:

$$U = \begin{pmatrix} U_1 \\ U_2 \\ \vdots \\ U_{nS} \end{pmatrix}, \quad \vec{F}^c = \begin{pmatrix} \vec{F}^c_1 \\ \vec{F}^c_2 \\ \vdots \\ \vec{F}^c_{nS} \end{pmatrix}, \quad \vec{F}^v = \begin{pmatrix} \vec{F}^v_1 \\ \vec{F}^v_2 \\ \vdots \\ \vec{F}^v_{nS} \end{pmatrix}, \quad Q = \begin{pmatrix} Q_1 \\ Q_2 \\ \vdots \\ Q_{nS} \end{pmatrix}, \tag{25}$$

where the conservative variables are $U_s = (\rho_s, \rho_s \vec{u}_s, \rho_s E_s, \rho_s e_{v_s})^T$ and

$$\vec{F}^c_s = \begin{pmatrix} \rho_s \vec{u}_s \\ \rho_s \vec{u}_s \otimes \vec{u}_s + \bar{\bar{I}} p_s \\ \rho_s H_s \vec{u}_s \\ \rho_s e_{v_s} \vec{u}_s + q_{vs} \end{pmatrix}, \vec{F}^v_s = \begin{pmatrix} \cdot \\ \bar{\bar{\tau}}_s \\ \bar{\bar{\tau}}_s \cdot \vec{u}_s + k_s \vec{\nabla} T_s \\ \cdot \end{pmatrix}, Q_s = \begin{pmatrix} \dot{w}_s \\ \vec{Q}_{u,s} + \vec{F}_{emf,s} \\ \vec{Q}_{u,s} \cdot \vec{u}_s + Q_{T,s} + W_{emf,s} \\ Q_{v,s} + \dot{w}_s e_{v_s} \end{pmatrix}, \tag{26}$$

where the vector of source terms, Q, includes the effects of finite rate chemistry, momentum, and energy exchange by collision and electromagnetic fields. The rate of production of mass by finite rate chemistry is given by $\dot{w}_s$, the rate of momentum exchange per unit volume by inter-species collision is $\vec{Q}_{u,s}$ and $Q_{T,s}$ is the rate of energy transfer per unit volume by inter-species collision. $Q_{v,s}$ is the rate of vibrational energy by inter-species collision between diatomic species. The momentum transfer per unit volume by force exerted on charged particles in the presence of electromagnetic fields is given by $\vec{F}_{emf,s}$, and the term $W_{emf,s}$ is the work done per unit volume in this process.

The plasma solver in SU^2 supports the following boundary conditions: Euler (flow tangency), and symmetry wall, no-slip (adiabatic, isothermal and catalytic), far-field and near-field characteristics and periodic boundary conditions. A detailed description of each of these terms and the catalytic wall boundary condition for this particular formulation of the governing equations wherein a full set of flow equations is solved for each species has been given by Lonkar et al.,[54] and Copeland et al.[16]

IV. Numerics

In this section, the main numerical algorithms of SU^2 will be described with a particular emphasis on the fluid dynamics solver, the grid adaptation capability, and the design methodology.

A. Space integration

Partial Differential Equations (PDEs) in SU^2 are discretized using a finite volume method[5, 28, 34–36, 53, 73, 87, 96] (a finite element method is also available) with a standard edge-based structure on a dual grid with control volumes constructed using a median-dual, vertex-based scheme as shown in Fig. 6. Median-dual control volumes are formed by connecting the centroids, face, and edge-midpoints of all cells sharing the particular node. The semi-discretized integral form of a typical PDE (like those described in the modeling section) is given by,

$$\int_{\Omega_i} \frac{\partial U}{\partial t}\, d\Omega + \sum_{j \in \mathcal{N}(i)} (\tilde{F}_{c_{ij}} + \tilde{F}_{v_{ij}})\Delta S_{ij} - Q|\Omega_i| = \int_{\Omega_i} \frac{\partial U}{\partial t}\, d\Omega + R_i(U) = 0, \tag{27}$$

where U is a vector of state variables, and $R_i(U)$ is the residual. $\tilde{F}_{c_{ij}}$ and $\tilde{F}_{v_{ij}}$ are the projected numerical approximations of the convective and viscous fluxes, respectively, and Q is a source term. ΔS_{ij} is the area of the face associated with the edge ij, Ω_i is the volume of the control volume and $\mathcal{N}(i)$ are the neighboring nodes to node i.

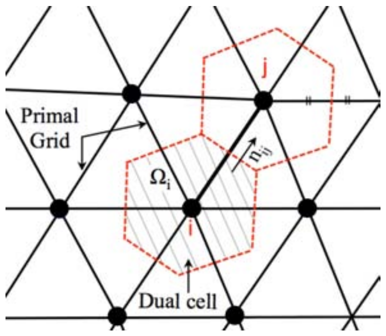


Figure 6. Schematic of the primal mesh and the control volume on a dual mesh.

The convective and viscous fluxes are evaluated at the midpoint of an edge. The numerical solver loops through all of the edges in the primal mesh in order to calculate these fluxes and then integrates them to evaluate the residual at every node in the numerical grid. In the following subsections, some of the most important numerical algorithms in SU^2 will be described.

1. *Integration of convective fluxes*

The convective fluxes can be discretized using central or upwind methods in SU^2. Several numerical schemes have been implemented (JST, Lax-Friedrich, Roe, AUSM, HLLC, Roe-Turkel), but this section will focus on two classic numerical schemes (Roe and JST).

The flux-difference-splitting scheme by Roe[75] evaluates the convective fluxes from flow quantities reconstructed separately on both sides of the face of the control volume from values at the surrounding nodes:

$$\tilde{F}_{c_{ij}} = \tilde{F}(U_i, U_j) = \left(\frac{\vec{F}_i^c + \vec{F}_j^c}{2}\right) \cdot \vec{n}_{ij} - \frac{1}{2} P|\Lambda|P^{-1}(U_i - U_j), \tag{28}$$

where $\vec{n}_{ij}$ is the outward unit normal associated with the face between nodes i and j, U_i is the vector of the conserved variables at point i and $\vec{F}_i^c$ is the convective flux at node i. P is the matrix of eigenvectors of the flux Jacobian matrix, constructed using the Roe averaged variables and projected in the $\vec{n}_{ij}$ direction, and $|\Lambda|$ is a diagonal matrix with entries corresponding to the absolute value of the eigenvalues of the flux Jacobian matrix. This discretization is first-order accurate in space. Second-order accuracy is easily achieved via reconstruction of variables on the cell interfaces by using a Monotone Upstream-centered Schemes for Conservation Laws (MUSCL) approach[91] with gradient limitation.

The JST scheme[41] uses a blend of two types of artificial dissipation that are computed using the differences in the undivided Laplacians (higher-order) of connecting nodes and the difference in the conserved variables (lower-order) on the connecting nodes. The two levels of dissipation are blended by using the typical pressure switch for triggering lower-order dissipation in the vicinity of shock waves. The final expression for the numerical flux using the JST method on unstructured meshes is:

$$\tilde{F}_{c_{ij}} = \tilde{F}(U_i, U_j) = \vec{F}^c\left(\frac{U_i + U_j}{2}\right) \cdot \vec{n}_{ij} - d_{ij}. \tag{29}$$

The artificial dissipation d_{ij} along the edge connecting nodes i and j can be expressed as

$$d_{ij} = \left(\varepsilon_{ij}^{(2)}(U_j - U_i) - \varepsilon_{ij}^{(4)}(\nabla^2 U_j - \nabla^2 U_i)\right)\varphi_{ij}\lambda_{ij}, \tag{30}$$

where the undivided Laplacians, local spectral radius, stretching in the grid and pressure switches are computed as

$$\begin{aligned}
\nabla^2 U_i &= \sum_{k\in\mathcal{N}(i)} (U_k - U_i), && (31)\\
\lambda_{ij} &= (|u_{ij}\cdot\vec{n}_{ij}| + c_{ij})\Delta S, \quad \lambda_i = \sum_{k\in\mathcal{N}(i)} \lambda_{ik}, && (32)\\
\varphi_{ij} &= 4\frac{\varphi_i\varphi_j}{\varphi_i + \varphi_j}, \quad \varphi_i = \left(\frac{\lambda_i}{4\lambda_{ij}}\right)^p, && (33)\\
\varepsilon_{ij}^{(2)} &= \kappa^{(2)} s_2\left(\left|\sum_{k\in\mathcal{N}(i)}(p_k - p_i)\right| \Big/ \sum_{k\in\mathcal{N}(i)}(p_k + p_i)\right), \quad \varepsilon_{ij}^{(4)} = s_4 \max\left(0, \kappa^{(4)} - \varepsilon_{ij}^{(2)}\right), && (34)
\end{aligned}$$

where $\mathcal{N}(i)$ represents the set of neighboring points to node i, p_i is the pressure at node i, s_2 and s_4 are stretching parameters and $\kappa^{(2)}$, $\kappa^{(4)}$ are adjustable parameters.

2. *Integration of viscous fluxes*

In order to evaluate the viscous fluxes using a finite volume method, flow quantities and their first derivatives are required at the faces of the control volumes. The values of the flow variables, including the velocity components, the dynamic viscosity μ and the heat conduction coefficient k, are averaged at the cell faces in SU^2.

The gradients of the flow variables are calculated using either a Green-Gauss or least-squares method at all grid nodes and then averaged to obtain the gradients at the cell faces. The following correction[95] is applied in order to reduce the truncation error of the scheme:

$$\nabla\phi\cdot\vec{n} = \frac{\phi_j - \phi_i}{|x_j - x_i|}\alpha_f + \frac{1}{2}(\nabla\phi|_i + \nabla\phi|_j)\cdot(\vec{n} - \alpha_f\vec{s}), \tag{35}$$

where $\vec{n}$ is the face normal, $\vec{s}$ is the normalized vector connecting the cell centroid across the face, $|x_j - x_i|$ is the distance between node i and j and α_f is chosen to be the dot product $\alpha_f = \vec{s}\cdot\vec{n}$. Again, the gradients $\nabla\phi|_i$ at node i are computed using either the Green-Gauss or least-squares theorems.

A Finite Element Method (FEM) is also available to numerically evaluate the Laplacian operator. Finite element methods are based upon approximations to a variational formulation of the problem. A variational formulation requires the introduction of a space of trial functions, $\mathcal{T} = \{V(t,\vec{x})\}$, and a space of weighting functions, $\mathcal{W} = \{W(t,\vec{x})\}$. The problem consists of finding $V(t,\vec{x})$ in $\mathcal{T}$ satisfying the problem boundary conditions, such that

$$\int_\Omega W^T\left(\nabla^2 V\right)\, d\Omega = 0. \tag{36}$$

To produce an approximate solution to the variational problem, a grid of finite elements is constructed on the domain, Ω. It will be assumed that the discretization employs p nodes. Finite-dimensional subspaces $\mathcal{T}^{(p)}$ and $\mathcal{W}^{(p)}$ of the trial and weighting function spaces, respectively, are defined by

$$\mathcal{T}^{(p)} = \left\{V^{(p)}(\vec{x})\,|\,V^{(p)} = \sum_{J=1}^{p} V_J N_J(\vec{x})\right\}, \;\mathcal{W}^{(p)} = \left\{W^{(p)}(\vec{x})\,|\,W^{(p)} = \sum_{J=1}^{p} a_J N_J(\vec{x})\right\}, \tag{37}$$

where V_J is the value of $V^{(p)}$ at node J. On the other hand, $a_1, a_2, \ldots, a_p$ are constant and $N_J(\vec{x})$ is the piecewise linear trial function associated with node J. We now apply the finite element approximation by discretizing the domain of the problem into elements and introducing functions which interpolate the solution over nodes that compose the elements. The Galerkin approximation is determined by applying the variational formulation of Eq. 36 in the following form: find $V^{(p)}$ in $\mathcal{T}^{(p)}$, satisfying the problem boundary conditions, such that

$$\int_\Omega N_I^T \left(\nabla^2 V\right) \, d\Omega = 0, \tag{38}$$

for $I = 1, 2, ..., p$. The form assumed for $V^{(p)}$ in Eq. 37 can now be inserted into the left hand side of Eq. 38 and the result can be written as

$$\int_\Omega N_I^T \left(\sum_{J=1}^{p} V_J \nabla^2 N_J \right) d\Omega = \sum_{J=1}^{p} V_J \left(\int_\Omega N_I^T \nabla^2 N_J \, d\Omega \right) = 0. \tag{39}$$

Applying the divergence theorem gives

$$\sum_{J=1}^{p} V_J \left(\int_\Gamma N_I^T \left(\nabla N_J \cdot \vec{\nu}\right) d\Gamma - \int_\Omega \nabla N_I^T \cdot \nabla N_J \, d\Omega \right) = 0, \tag{40}$$

where $\vec{\nu}$ is the outward unit normal associated with the control volume surface and the boundary integral disappears unless we are computing a boundary element with non-homogeneous Neumann conditions (I is an exterior node). The result at a typical interior node I is

$$\sum_{E \in I} \sum_{J \in E} V_J \left(\int_{\Omega_E} \nabla N_I^T \cdot \nabla N_J \, d\Omega \right) = 0, \tag{41}$$

where the first summation extends over the elements E in the numerical grid which contain node I and the second summation extends over nodes J of the elements E. Ω_E is the portion of Ω which is represented by element E.

3. Source term integration

Source terms are approximated using piecewise constant reconstruction within each of the finite volume cells. The source terms plays a fundamental role in plasma simulations (chemical reactions), free-surface problems (gravity effects), or in the formulation of turbulence and transition models.

B. Time integration

1. Steady simulations

It is well known that Eq. 27 has to be valid over the whole time interval, so one has to make the choice of evaluating $R_i(U)$ either at time t^n (explicit methods) or t^{n+1} (implicit methods). Focusing on the implicit integration (SU^2 also has a Runge-Kutta explicit method), the easiest way to discretize the system is by using an implicit Euler scheme which can be written as

$$\int_{\Omega_i} \frac{\partial U}{\partial t} \, d\Omega + R_i(U) \approx |\Omega_i| \frac{\mathrm{d}U_i}{\mathrm{d}t} + R_i(U) = 0 \quad \rightarrow \quad \frac{|\Omega_i^n|}{\Delta t_i^n} \Delta U_i^n = -R_i(U^{n+1}), \tag{42}$$

where $\Delta U_i^n = U_i^{n+1} - U_i^n$. However, the residuals at time $n+1$ are unknown, and linearization about t^n is needed:

$$R_i(U^{n+1}) = R_i(U^n) + \frac{\partial R_i(U^n)}{\partial t} \Delta t_i^n + \mathcal{O}(\Delta t^2) = R_i(U^n) + \sum_{j \in \mathcal{N}(i)} \frac{\partial R_i(U^n)}{\partial U_j} \Delta U_j^n + \mathcal{O}(\Delta t^2). \tag{43}$$

Finally, the following linear system should be solved to find the solution update (ΔU_i^n),

$$\left(\frac{|\Omega_i|}{\Delta t_i^n} \delta_{ij} + \frac{\partial R_i(U^n)}{\partial U_j} \right) \cdot \Delta U_j^n = -R_i(U^n), \tag{44}$$

where if a flux $\tilde{F}_{ij}$ has a stencil of points $\{i, j\}$, then contributions are made to the Jacobian at four points:

$$\frac{\partial R}{\partial U} := \frac{\partial R}{\partial U} + \begin{bmatrix} \ddots & & & & \\ & \frac{\partial \tilde{F}_{ij}}{\partial U_i} & \cdots & \frac{\partial \tilde{F}_{ij}}{\partial U_j} & \\ & \vdots & \ddots & \vdots & \\ & -\frac{\partial \tilde{F}_{ij}}{\partial U_i} & \cdots & -\frac{\partial \tilde{F}_{ij}}{\partial U_j} & \\ & & & & \ddots \end{bmatrix}. \tag{45}$$

Note that, despite implicit schemes being unconditionally stable in theory, a specific value of Δt_i^n is needed to relax the problem. SU^2 uses a local-time-stepping technique to accelerate convergence to a steady state. Local-time-stepping allows each cell in the mesh to advance at a different local time step. Calculation of the local time step requires the estimation of the eigenvalues and first-order approximations to the Jacobians at every node i according to

$$\Delta t_i = N_{CFL} \min\left(\frac{|\Omega_i|}{\lambda_i^{conv}}, \frac{|\Omega_i|}{\lambda_i^{visc}}\right), \tag{46}$$

where N_{CFL} is the Courant-Friedrichs-Lewy (CFL) number, $|\Omega_i|$ is the volume of the cell i and λ_i^{conv} is the integrated convective spectral radius[22] computed as

$$\lambda_i^{conv} = \sum_{j \in \mathcal{N}(i)} (|\vec{u}_{ij} \cdot \vec{n}_{ij}| + c_{ij})\Delta S, \tag{47}$$

where $\vec{u}_{ij} = (\vec{u}_i + \vec{u}_j)/2$, and $c_{ij} = (c_i + c_j)/2$ denote the velocity and the speed of sound at the cell face. $\vec{n}_{ij}$ denotes the normal direction of the control surface and ΔS, its area. On the other hand, the viscous spectral radius λ_i^{visc} is computed as

$$\lambda_i^{visc} = \sum_{j \in \mathcal{N}(i)} C \frac{\mu_{ij}}{\rho_{ij}} S_{ij}^2, \tag{48}$$

where C is a constant, μ_{ij} is the sum of the laminar and eddy viscosities in a turbulent calculation and ρ_{ij} is the density evaluated at the midpoint of the edge ij.

2. Unsteady simulations

A dual time-stepping strategy[33,40] has been implemented to achieve high-order accuracy in time. In this method, the unsteady problem is transformed into a steady problem at each physical time step which can then be solved using all of the well known convergence acceleration techniques for steady problems. The current implementation of the dual-time stepping approach solves the following problem

$$\frac{\partial U}{\partial \tau} + R^*(U) = 0, \tag{49}$$

where

$$R^*(U) = \frac{3}{2\Delta t}U + \frac{1}{|\Omega|^{n+1}}\left(R(U) - \frac{2}{\Delta t}U^n|\Omega|^n + \frac{1}{2\Delta t}U^{n-1}|\Omega|^{n-1}\right), \tag{50}$$

where Δt is the physical time step, τ is a fictitious time used to converge the steady state problem, $R(U)$ denotes the residual of the governing equations, and $U = U^{n+1}$ once the steady problem is satisfied.

3. Linear solvers

The SU^2 framework includes the implementation of several linear solvers for solving Eq. 44. Specifically, the following methods are available:

- The Lower-Upper Symmetric-Gauss-Seidel (LU-SGS) method.[42,45,99] This is a stationary iterative method that is based on a measurement of the error in the result (the residual) which is used to form a "correction equation".

- The Generalized Minimal Residual (GMRES) method,[77] which approximates the solution by the vector in a Krylov subspace with minimal residual. The Arnoldi iteration is used to find this vector.
- The Biconjugate Gradient Stabilized (BiCGSTAB) method,[90] also a Krylov subspace method. It is a variant of the biconjugate gradient method (BiCG) and has faster and smoother convergence properties than the original BiCG.

C. Convergence acceleration techniques

In this section some of the most important convergence acceleration techniques of SU^2 will be described: Multigrid and linelet techniques, followed by Roe-Turkel preconditioning for low Mach number flows. It is fundamental to note that, due to the modular structure of SU^2, these methods are shared by all the solvers in the code.

1. *Nonlinear multigrid method*

The multigrid method generates effective convergence at all length scales of a problem by employing a sequence of grids of varying resolution. Simply stated, the main idea is to accelerate the convergence of the numerical solution of a set of equations by computing corrections to the fine-grid solutions on coarser grids and applying this idea recursively.[7, 39, 58, 59, 66] It is well know that, owing to the nature of most iterative methods/relaxation schemes, high-frequency errors are usually well damped, but low-frequency errors (global error spanning the solution domain) are less damped by the action of iterative methods that have a stencil with a local area of influence.

The basic methodology is described below. Consider the nonlinear problem $L(u) = f$ defined in a domain Ω, and denote its discretization on a fine grid with spacing h as

$$L_h(u_h) = f_h, \quad \text{in } \Omega_h, \tag{51}$$

where $L_h(\cdot)$ is a nonlinear discrete operator defined in Ω_h. The starting point is the definition of a suitable smoother (e.g. LU-SGS, GMRES, etc.) and, after a small number of iterations of this method (possibly a single one, instead of fully solving the discrete equation), an approximate solution $\bar{u}_h$, and residual r_h, are obtained on the fine grid. The resulting equation on the fine grid can be written as

$$L_h(\bar{u}_h) - f_h = r_h. \tag{52}$$

Subtracting Eq. 52 from Eq. 51, we obtain the following expression to be approximated on a coarse grid:

$$L_h(u_h) - L_h(\bar{u}_h) = -r_h, \tag{53}$$

where the exact solution u_h can be expressed as the approximate solution $\bar{u}_h$ plus a correction c_h yielding:

$$L_h(\bar{u}_h + c_h) - L_h(\bar{u}_h) = -r_h. \tag{54}$$

Note that no assumptions about the linearity of the operator $L(\cdot)$ (or its discrete version) are made. As we stated before, the objective is to write Eq. 54 on a coarse grid of spacing H. In order to do that, two types of restriction operators will be defined: I_h^H, the restriction operator that interpolates the residual from the fine grid h to the coarse grid H (in a conservative way), and $\bar{I}_h^H$, which simply interpolates the fine-grid solution onto the coarse grid. Formulating Eq. 54 on the coarse level by replacing $L_h(\cdot)$ with $L_H(\cdot)$, $\bar{u}_h$ with $\bar{I}_h^H \bar{u}_h$, and r_h with $I_h^H r_h$, we obtain the FAS equation:

$$L_H(\bar{I}_h^H \bar{u}_h + c_H) - L_H(\bar{I}_h^H \bar{u}_h) = -I_h^H r_h. \tag{55}$$

In this last expression, the approximate solution on the coarse grid is denoted as $\bar{u}_H := \bar{I}_h^H \bar{u}_h + c_H$ and the residual r_h can be written as $L_h(\bar{u}_h) - f_h$. Finally we obtain the following useful equation on the coarse level:

$$L_H(\bar{u}_H) = L_H(\bar{I}_h^H \bar{u}_h) - I_h^H(L_h(\bar{u}_h) - f_h) = f_H + \tau_h^H, \quad \text{in } \Omega_H, \tag{56}$$

where the source term on the coarse levels is interpolated $f_H = I_h^H f_h$ (not computed), and a new variable $\tau_h^H = L_H(\bar{I}_h^H \bar{u}_h) - I_h^H(L_h \bar{u}_h)$ is defined as the fine-to-coarse defect or residual correction. Note that without the τ_h^H term, the coarse grid equation is the original system represented on the coarse grid.

The next step is to update the fine grid solution. For that purpose, the coarse-grid correction c_H (which in principle is smooth because of the application of the smoothing iteration) is interpolated back on to the fine grid using the following formula

$$\bar{u}_h^{new} = \bar{u}_h^{old} + I_H^h(\bar{u}_H^{new} - \bar{I}_h^H \bar{u}_h^{old}), \tag{57}$$

where I_H^h is a prolongation operator that interpolates the coarse-grid correction onto the fine grid. Note that we interpolated the correction and not the coarse-grid solution itself. In this brief introduction to the method, only two grids have been considered. In real problems, however, the algorithm is applied in a recursive way using different grid level sizes to eliminate the entire spectrum of frequencies of the numerical error.

Because of their structured-grid heritage, multigrid methods have traditionally been developed from a geometric point of view. The agglomeration multigrid technique developed in SU^2 is based on the agglomeration of fine-grid control volumes to create a coarse-grid structure. There are multiple criteria that should be observed to increase the quality of the agglomerated grid. But, in all cases, the basic idea is to maintain as much as possible the quality of the finer grid in the agglomerated levels. In other words, if the finer grid has good orthogonality and stretching properties, the coarse levels should preserve those properties. In particular, three aspects have been carefully treated: selection of the seed control volume, preservation of the stretching and volume constraints, agglomeration of indirect neighbors in structured grids.

In order to choose the seed control volume in the agglomeration process, the first step is the creation of an agglomeration priority list based on geometrical properties of the grid surfaces (i.e., a vertex has the highest priority, then common edges between surfaces and finally solid surfaces). It is important to note that far-field, inlet or outlet surfaces always have the lowest priority among the surfaces. Once the geometrical division is done, the control volumes with the same geometrical priority should be divided into different groups depending on the number of neighbors that have been previously agglomerated.

The algorithm will choose the first control volume with the highest priority as a seed point. Once the seed point is agglomerated with its neighbors, it is removed from the queue, and its neighbors increase in priority. Apart from the seed selection criteria, there are some situations in which a control volume cannot be agglomerated with its neighbors. In particular, three basic constraints have been implemented in the agglomeration process:

- The stretching of the agglomerated control volume is limited by a percentage of the stretching of the original fine grid control volume.
- The volume of the agglomerated element should be less than a percentage of the total volume of the computational grid.
- There is a maximum number of control volumes that can be agglomerated to the original element.

The last two criteria are easy to implement and they do not required further description. With respect to the stretching factor, it is computed by comparing the maximum and minimum distance between control points connected to the reference control volume. The stretching factor should be computed on the finer grid and copied to all coarse levels as reference value in the agglomeration process.

To maintain the quality for structured grids (finest level), the agglomeration of the indirect neighbors (control volumes that have a vertex in common with the seed control volume) is critical. In this particular algorithm the structured part of the grid is identified, and the indirect neighbors of an element will be added to the new agglomerated control volume. Information about the control volume relationship structure is copied from the finest grid to the coarse levels.

2. *Linelet preconditioning*

Preconditioning is the application of a transformation to the original system that makes it more suitable for numerical solution.[71] In particular, a linelet preconditioner has been implemented to improve the convergence rate of the Krylov subspace linear solvers. The key[59,83] is to construct lines in the mesh in the direction normal to the grid stretching. The preconditioning matrix is built by assembling the diagonal entries of the system matrix and the non-diagonal entries of the edges that belong to these linelets. If the appropriate numbering is used, a block tridiagonal matrix is obtained, and the preconditioned system can be directly inverted using the Thomas algorithm. Note that in those zones where linelets are not defined, a Jacobi preconditioner is used.

In summary, in order to solve the system $Ax = b$ using a preconditioned Krylov subspace method, it is necessary to solve (twice in each iteration) a system of equations of the form $Pz = r$, where P is the linelet preconditioner. The steps of the algorithm are:

1. Build grid lines (linelets) in the direction normal to the grid stretching.
2. Build the preconditioner matrix: assemble the diagonal and the non-diagonal entries (linelets) of the Jacobian matrix.
3. Nodal renumbering following the linelets to obtain a tridiagonal structure for the preconditioner.
4. Use the preconditioned formulation of the iterative scheme.
5. Solve the tridiagonal system for the linelets using the Thomas algorithm (direct method).

The linelet creation begins with the identification of all the points that are on the solid surface of the geometry (where boundary layers and/or wakes are likely to exist) and the computation of an edge weight for each vertex on the surface. This weight is computed as

$$w_{ij} = \frac{1}{2}\Delta S_{ij}\left(\frac{1}{|\Omega_i|} + \frac{1}{|\Omega_j|}\right), \tag{58}$$

where ΔS_{ij} is the area of the face that separates nodes i and j and $|\Omega_i|$ is the volume of the control volume associated with each node. The line is built by adding to the original vertex the one which is most strongly connected to it (maximum value of the weight). This new vertex is added only if it has not already been added to another linelet and if the weight is greater than a certain quantity that is used to mark the termination of the linelet. When an entire line is completed, the procedure is repeated starting with another vertex on the surface.

Once the list of linelets is built, the nodal points must be renumbered following the linelets to obtain the desired tridiagonal structure of the preconditioner. First, the nodes of a single linelet are renumbered from one end to the other, then a second linelet is renumbered, and so on until all linelets have been covered. Finally, the rest of the points also have to be renumbered to accommodate the renumbering of the vertices that belong to the collection of linelets that have been identified.

The final step is to solve the system $Pz = r$ using the Thomas' algorithm. The preconditioner matrix is decomposed into upper- and lower-triangular matrices U and L. Once those matrices have been obtained, the solution of the system $Pz = r$ is computed by performing the following substitutions:

$$y_i = r_i - L_i y_{i-1}, (y_1 = r_1),\ i = 2, ..., n; \tag{59}$$
$$z_i = U^{-1}(y_i - F_i z_{i+1}), (z_{n+1} = 0),\ i = n, ..., 1. \tag{60}$$

It is important to highlight that the LU decomposition is done only once and can be done on a per-linelet basis. On the other hand, an LU factorization is performed at each block of the implicit block matrix to compute its inverse. Owing to the particular renumbering chosen where a linelet is not defined, a Jacobi preconditioner is recovered by default.

3. Roe-Turkel low Mach number preconditioning

Numerical discretization of the governing fluid dynamic equations using a conservative formulation often results in excess artificial viscosity at low Mach numbers. This degrades the performance of a compressible solver in regions of low Mach number flow. Preconditioning techniques such as Roe-Turkel[89] have been developed for solving nearly incompressible flow problems using the same numerical methods developed for compressible flows. This can be particularly useful when only part of a flow field is essentially incompressible. For example, flow over a multi-element airfoil at high angles of attack has regions of both compressible and incompressible flow. Furthermore, flows around re-entry vehicles moving at hypersonic speeds are often chemically dissociated into several species moving hypersonically, except the electron species which, owing to their small mass, have a very small (often less than 0.01) Mach number.

The poor convergence of the standard discretization methods at low Mach numbers is due to an incorrect scaling of the artificial viscosity as the Mach number tends to zero. Turkel proposed[88,93] a correction to the convective flux discretization method which involves modifying the artificial viscosity for correct scaling at low Mach numbers using a preconditioning matrix. The Roe-Turkel technique is a modification to the standard Roe scheme involving a preconditioner activated in regions of low Mach numbers. Eq. 61 shows the modification of the artificial viscosity with a preconditioning matrix P_c.

$$\tilde{F}_{RT}(U_i, U_j) = \left(\frac{\vec{F}_i^c + \vec{F}_j^c}{2} \right) \cdot \vec{n}_{ij} - \frac{1}{2} P_c^{-1} |P_c \hat{A}| (U_i - U_j), \tag{61}$$

where it is important to highlight that this technique does not hamper the time accuracy of results and can be used for unsteady problems.

D. Mesh manipulation

1. *Dynamic meshes for unsteady simulations*

Unsteady simulations with surfaces in motion can be performed by solving the ALE form of the governing equations along with a suitable method for moving the surfaces and mesh while computing mesh velocities. A straightforward method is the use of rigid motion of the entire domain, where there is no relative motion between individual grid nodes. Rigid rotational and translational motion for a mesh node with each physical time step can be generally described by

$$\vec{x}^{\,n+1} = \mathfrak{R}\,\vec{x}^{\,n} + \Delta\vec{x}, \tag{62}$$

where $\vec{x}^{\,n} = \{x, y, z\}^\mathsf{T}$ is the current node position in Cartesian coordinates, $\vec{x}^{\,n+1}$ is the updated node location at the next physical time instance, $\Delta\vec{x}$ is a vector describing the translation of the nodal coordinates between time steps, and in three dimensions, the rotation matrix, $\mathfrak{R}$, is given by

$$\mathfrak{R} = \begin{bmatrix} \cos\theta_y \cos\theta_z & \sin\theta_x \sin\theta_y \cos\theta_z - \cos\theta_x \sin\theta_z & \cos\theta_x \sin\theta_y \cos\theta_z + \sin\theta_x \sin\theta_z \\ \cos\theta_y \sin\theta_z & \sin\theta_x \sin\theta_y \sin\theta_z + \cos\theta_x \cos\theta_z & \cos\theta_x \sin\theta_y \sin\theta_z - \sin\theta_x \cos\theta_z \\ -\sin\theta_y & \sin\theta_x \cos\theta_y & \cos\theta_x \cos\theta_y \end{bmatrix} \tag{63}$$

with $\Delta\vec{\theta} = \{\theta_x, \theta_y, \theta_z\}^\mathsf{T}$ being equal to the change in angular position of the nodal coordinates about a specified rotation center between time $t^{\,n+1}$ and $t^{\,n}$. Note that this matrix is formed by assuming positive, right-handed rotation first about the x-axis, then the y-axis, and finally the z-axis. The general form of Eqn. 62 supports multiple types of motion, including constant rotational or translational rates, pitching, or plunging. With each physical time step, the values of $\Delta\vec{\theta}$ and $\Delta\vec{x}$ are computed and Eqn. 62 is applied at each node of the mesh.

For all types of rigid mesh motion, the local grid velocity at a node, $\vec{u}_x$, which is needed for solving the ALE form of the governing equations, can be computed by storing the node locations at prior time instances and using a finite differencing approximation that is consistent with the chosen dual time-stepping scheme. For second-order accuracy in time, the mesh velocities are given by

$$\vec{u}_x \approx \frac{3\vec{x}^{\,n+1} - 4\vec{x}^{\,n} + \vec{x}^{\,n-1}}{2\Delta t}, \tag{64}$$

where Δt is the physical time step.

Mesh motion can also be accomplished by first moving the surface boundaries in some specified manner and then deforming the volume mesh to conform to the new surface position with each time step. The volumetric deformation procedure is based on a classical spring method, where the key element is the definition of a stiffness matrix k_{ij}, that connects the two ends of a single bar (mesh edge). Equilibrium of forces is then imposed at each mesh node

$$\left(\sum_{j \in \mathcal{N}(i)} k_{ij} \vec{e}_{ij} \vec{e}_{ij}^\mathsf{T} \right) \vec{u}_i = \sum_{j \in \mathcal{N}(i)} k_{ij} \vec{e}_{ij} \vec{e}_{ij}^\mathsf{T} \vec{u}_j, \tag{65}$$

where the displacement $\vec{u}_i$ is unknown and is computed as a function of the known surface displacements $\vec{u}_j$. $\mathcal{N}(i)$ is the set of neighboring points to node i and $\vec{e}_{ij}$ the unit vector in the direction connecting both points. The system of equations is solved iteratively using the same linear solvers described above. This technique for mesh deformation is also during optimal shape design (described below) in order to modify the volume mesh between major design iterations after an optimizer provides new values for the design variables that control the geometry (surface shape).

2. Mesh adaptation

An adaptive mesh refinement procedure is included in SU^2. Using this method, an existing mesh can be dynamically modified by the code to improve the accuracy of the solution. This is done to improve the accuracy of a solution without an excessive increase in computational effort. The implemented procedure[6] uses anisotropic adaptation of the grid based on an edge binary system that identifies the right division for each element. This methodology is important to maintain the coherence in the division of the common faces between control volumes. Some of the main important characteristics of this adaptation strategy are:

- Robust adaptation. The method should allow multiple adaptation cycles, using classical two-dimensional and three-dimensional finite volume elements (tetrahedra, hexahedra, pyramids and wedges).
- Fully automated and easy to use.
- No “hanging” nodes, the final grid should be conforming.

A method based on a flexible element division has been implemented in addition to the tetrahedral and hexahedral division methods which might not give optimum results in some cases. The tetrahedral division procedure is based on the detection of edge division patterns that are prescribed in the code, however, the anisotropic division of tetrahedra can significantly deteriorate the quality of the mesh. This is a serious problem when directional flow-field features are present.

The main challenge of hexahedral division methodologies is the so-called refinement propagation problem (the buffer zone between an adapted element and the non-adapted grid is greater than one cell). This occurs when more than one set of edges in the same hexahedron are marked non-uniformly and “hanging” nodes are not allowed.

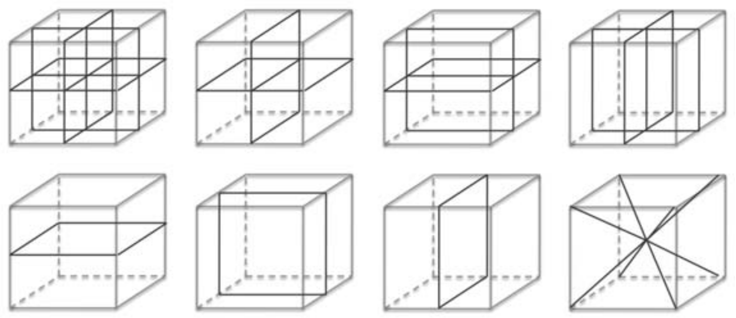


Figure 7. Allowed hexahedral divisions, the last division requires a new vertex in the middle of the hexahedron.

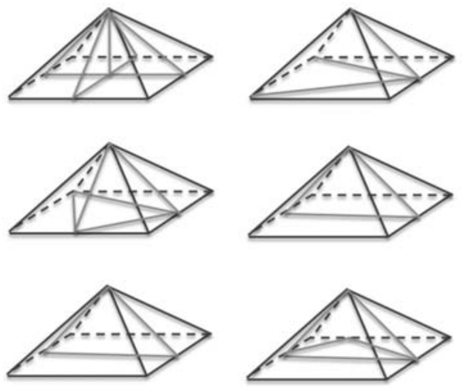


Figure 8. Allowed pyramids divisions, a total number of 16 combinations are possible.

To prevent the refinement propagation, SU^2 uses a flexible element division technique.[6] Once an edge-based structure is in place, the allowed hexahedral division is identified. Fig. 7 shows the seven divisions/combinations implemented in the code. The last division is a special division with a vertex inserted in the middle of a marked hexahedron. The special hexahedral division into pyramids is required to connect the adapted with the non-adapted grid, furthermore, each pyramid will be divided into tetrahedrons (see Fig. 8).

E. Adjoint formulation and sensitivity computation

In gradient-based optimization techniques (including grid adaptation), the goal is to minimize a suitable cost or objective function (drag, lift, etc.) with respect to a set of design variables. Minimization is achieved by means of an iterative process that requires the computation of the gradients or sensitivity derivatives of the cost function with respect to the design variables.

Gradients of an objective function can be computed in a variety of ways, some of the most popular being the adjoint methods,[2,12,32,38,63,72] due, among other factors, to their ability of computing these derivatives at a cost comparable to solving the state PDEs (Partial Differential Equations). Adjoint methods are conventionally divided into continuous and discrete.[37,60,64] In the continuous approach,[4,8,38,43,44,49] the adjoint equations are derived from the governing PDE and then subsequently discretized, whereas in the discrete approach[64,86] the adjoint equations are directly derived from the discretized governing equations. In SU^2 both methodologies have been implemented.

1. *Continuous adjoint methodology*

One of the key features of SU^2 is its capability to perform optimal shape design based on the continuous adjoint methodology. In this section, the main steps of the continuous adjoint methodology applied to the Navier-Stokes equations will be shown.

Optimal shape design in systems governed by partial differential equations is formulated on a fluid domain Ω, delimited by disconnected boundaries divided into an inlet/outlet and one or more solid wall boundaries S. From now on we will restrict ourselves to the analysis of optimization problems involving functionals J defined on the solid wall S, whose value depends on the flow variables U obtained from the solution of the fluid flow equations. In this context, the generic optimization problem can be succinctly stated as follows: find $S^{min} \in \mathcal{S}_{ad}$ such that

$$J(S^{min}) = \min_{S^{min} \in \mathcal{S}_{ad}} J(S), \tag{66}$$

where $\mathcal{S}_{ad}$ is the set of admissible boundary geometries and

$$J(S) = \int_S j(\vec{f}, T, \vec{n})\, ds, \tag{67}$$

is the objective function, where $j(\vec{f}, T, \vec{n})$, is a smooth function which depends on $\vec{n}$ (inward-pointing unit vector normal to S), the temperature T on the surface and the vector $\vec{f} = (f_1, f_2, f_3) = P\vec{n} - \bar{\sigma} \cdot \vec{n}$, where P is the pressure of the fluid, and $\bar{\sigma}$ the second order tensor of viscous stresses.

As usual in the adjoint approach the flow equations are incorporated to the cost function as constraints by means of a Lagrange multiplier for each equation, $\Psi^{\mathsf{T}} = (\psi_1, \psi_2, \psi_3, \psi_4, \psi_5)$. In this way, the Lagrangian reads

$$\mathcal{J}(S) \quad = \quad \int_S j(\vec{f}, T, \vec{n})\, ds + \int_\Omega \left(\Psi^{\mathsf{T}} R(U)\right)\, d\Omega.$$

where $R = R(U)$ are the Navier-Stokes equations. Let us consider an arbitrary (but small) perturbation of the boundary S which, without loss of generality, can be parameterized by an infinitesimal deformation of size δS along the normal direction to the surface S. The new surface obtained after the deformation is then given by $S' = \{\vec{x} + \delta S\, \vec{n},\, \vec{x} \in S\}$ where, for small deformations, the following holds[82]

$$\delta\vec{n} = -\nabla_S(\delta S), \quad \delta(ds) = -2H_m \delta S\, ds, \tag{68}$$

where H_m is the mean curvature of S computed as $(\kappa_1 + \kappa_2)/2$, and κ_1, κ_2 are curvatures in two orthogonal directions on the surface. Here ∇_S represents the tangential gradient operator on S. Note that $\nabla_S(\delta S)$ is a tangent vector to S in $\mathbb{R}^3$ with null component normal to S.

Assuming a regular flow solution U and a smooth boundary S, the variation of the functional J due to the deformation can be evaluated as

$$\delta\mathcal{J} \quad = \quad \int_S \delta j(\vec{f}, T, \vec{n})\, ds + \int_{\delta S} j(\vec{f}, T, \vec{n})\, ds + \int_\Omega \left(\Psi^{\mathsf{T}} \delta R(U)\right) d\Omega, \tag{69}$$

where δR represents the variations of R. Using the convention of summation of repeated indexes, $i = 1, 2, 3$, the two first terms in the previous equation read

$$\begin{aligned}
\int_S \delta j(\vec{f}, T, \vec{n}) \, ds &= \int_S \left(\frac{\partial j}{\partial f_i} \delta f_i + \frac{\partial j}{\partial T} \delta T - \frac{\partial j}{\partial \vec{n}} \cdot \nabla_S(\delta S) \right) ds \\
&= \int_S \left(\frac{\partial j}{\partial \vec{f}} \cdot (\delta P \vec{n} - \delta \bar{\sigma} \cdot \vec{n}) + \frac{\partial j}{\partial T} \delta T - \left(\frac{\partial j}{\partial \vec{n}} + \frac{\partial j}{\partial \vec{f}} P - \frac{\partial j}{\partial \vec{f}} \cdot \bar{\sigma} \right) \cdot \nabla_S(\delta S) \right) ds, \quad (70) \\
\int_{\delta S} j(\vec{f}, T, \vec{n}) \, ds &= \int_S \left(\frac{\partial j}{\partial f_i} \partial_n f_i + \frac{\partial j}{\partial T} \partial_n T - 2 H_m j \right) \delta S \, ds \quad (71)
\end{aligned}$$

Note that in Eq. 70 we have written the variation $\delta \vec{f}$ in terms of δP and $\delta \bar{\sigma}$ and used Eq. 68 for $\delta \vec{n}$. The variations $\delta P \vec{n} - \delta \bar{\sigma} \cdot \vec{n}$ and δT appearing in Eq. 70 can be computed from the following linearized system

$$\left\{ \begin{array}{ll} \delta R(U) = \dfrac{\partial R}{\partial U} \delta U = 0 & \text{in } \Omega, \\ \delta \vec{v} = -\partial_n \vec{v} \, \delta S & \text{on } S, \\ \partial_n (\delta T) = (\nabla T) \cdot \nabla_S(\delta S) - \partial_n^2 T \delta S & \text{on } S, \end{array} \right. \quad (72)$$

The domain integrals in Eq. 69 are eliminated by using integration by parts and introducing the associated adjoint operators. Integrating by parts also creates additional boundary terms, which are combined with the boundary terms in Eq. 69 depending on $\delta P \vec{n} - \delta \sigma \cdot \vec{n}$ and δT, yielding the boundary conditions for the adjoint operators. Following this procedure, the continuous adjoint system becomes

$$\left(\frac{\partial R}{\partial U} \right)^{\mathsf{T}} \Psi = 0 \quad \text{in } \Omega. \quad (73)$$

Analogously, all boundary terms without explicit dependence on δS can be eliminated by considering the following choice of boundary conditions for the adjoint variables

$$\left\{ \begin{array}{ll} \varphi_i = \frac{\partial j}{\partial f_i} & \text{on } S, \\ \partial_n \psi_5 = \frac{1}{\mu^*_{tot} C_p} \left(\frac{\partial j}{\partial T} \right) & \text{on } S. \end{array} \right. \quad (74)$$

where μ_{tot} and μ^*_{tot} have been defined in Eq. 3. Finally it is possible to write the variation of the functional $\delta \mathcal{J}$ as

$$\delta \mathcal{J} = -\int_S h \delta S \, ds = \int_S (\vec{n} \cdot \bar{\Sigma}^{\varphi} \cdot \partial_n \vec{v} - \mu^*_{tot} C_p \nabla_S \psi_5 \cdot \nabla_S T) \, \delta S \, ds, \quad (75)$$

where h is the shape sensitivity which does not depend on the variation of the flow variables and $\bar{\Sigma}^{\varphi}$ is computed as

$$\bar{\Sigma}^{\varphi} = \mu_{tot} \left(\nabla \vec{\varphi} + \nabla \vec{\varphi}^{\mathsf{T}} - \mathbf{I}_d \frac{2}{3} \nabla \cdot \vec{\varphi} \right). \quad (76)$$

2. *Discrete adjoint methodology*

A discrete adjoint methodology, based on Automatic Differentiation,[17, 25] has been also implemented in SU^2. In this case, the sensitivity of the numerical residual R to a parameter α can be written as:

$$\frac{dR}{d\alpha} = \frac{\partial R}{\partial U} \frac{dU}{d\alpha} + \frac{\partial R}{\partial \alpha} = 0, \quad (77)$$

rearranging this last equation gives:

$$\frac{dU}{d\alpha} = -\left(\frac{\partial R}{\partial U} \right)^{-1} \frac{\partial R}{\partial \alpha}. \quad (78)$$

In a similar way the sensitivity of the objective function J to the parameter α can be written as

$$\frac{dJ}{d\alpha} = \frac{\partial J}{\partial U} \frac{dU}{d\alpha} + \frac{\partial J}{\partial \alpha}, \quad (79)$$

where using the result from Eq. 78 gives

$$\frac{dJ}{d\alpha} = -\frac{\partial J}{\partial U}\left(\frac{\partial R}{\partial U}\right)^{-1}\frac{\partial R}{\partial \alpha} + \frac{\partial J}{\partial \alpha}. \tag{80}$$

Finally, the adjoint equation is defined as:

$$\left(\frac{\partial R}{\partial U}\right)^{T}\Psi = -\left(\frac{\partial J}{\partial U}\right)^{T}, \tag{81}$$

and the variation of the objective function can be written as

$$\frac{dJ}{d\alpha} = \Psi^{T}\frac{\partial R}{\partial \alpha} + \frac{\partial J}{\partial \alpha}. \tag{82}$$

From the implementation point of view, the discrete adjoint solver in SU^2 uses the same class structure as the existing continuous adjoint and builds the required Jacobian matrix by mirroring the solution method of the direct problem.

The flow the residual is calculated by first looping over each edge to find the contribution from upwinding and then looping over the boundary nodes to add the appropriate boundary condition contributions. Hence, in the discrete adjoint, the contributions to the Jacobian matrix from the fluxes across edges are first calculated, followed by the contributions from the boundary nodes.

The derivatives of the fluxes, used in both the loop over edges and the boundary condition, are found by applying Automatic Differentiation. A differentiated version of the upwind routine was created using Python and Tapenade,[26] first extracting and converting the C++ code to C using Python, then transforming this code using Tapenade, and finally converting back to C++ and inserting the differentiated routine into SU^2, again using Python.

Once the Jacobian, which is assembled in a transposed state, and objective function sources are constructed the linear system can be solved via an iterative procedure.

F. Goal-oriented grid adaptation

The error estimate of integral outputs of partial differential equations can be used as goal-oriented grid adaptation indicators.[24,69] These techniques produce good (and even optimal) numerical grids for the accurate estimation of an output functional. To illustrate the main idea behind this technique, suppose that a nonlinear function $J(U)$ (e.g. heat flux, temperature, or pressure distributions on a body surface) is to be evaluated, where U is the exact solution of a set of nonlinear equations $R(U) = 0$.

Given an approximate solution, $\bar{U}$, we define u as the error of the solution, $u = \bar{U} - U$, and linearize both the nonlinear equation and the functional:

$$R(\bar{U}) = R(U+u) \approx \frac{\partial R}{\partial U}u, \tag{83}$$

and

$$J(\bar{U}) = J(U+u) \approx J(U) + \frac{\partial J}{\partial U}u. \tag{84}$$

This can be re-written as $Au \approx f$, where $A = \partial R/\partial U$, $f = R(\bar{U})$, and $J(U) \approx J(\bar{U}) - g^T u$ where $g^T = \partial J/\partial U$. If u satisfies the primal equation and v satisfies the dual or adjoint equation then $A^T v = g$. Hence,

$$J(U) \approx J(\bar{U}) - v^T f \approx J(\bar{U}) - v^T R(\bar{U}), \tag{85}$$

where $J(\bar{U}) - v^T R(\bar{U})$ is a more accurate estimate for $J(U)$ than $J(\bar{U})$. This computable correction $v^T R(\bar{U})$ is the sensor for our goal-oriented adaptation.

G. Design variable definition

Using the continuous adjoint methodology, SU^2 can compute the variation of an objective function with respect to infinitesimal surface shape deformations in the direction of the local surface normal at points on the design surface. While it is possible to use each surface node in the computational mesh as a design variable capable of deformation, this approach is not often pursued in practice.

A more practical choice is to compute the surface sensitivities at each mesh node on the design surface and then to project this information into a design space made up of a smaller set (possibly a complete basis) of design variables. This procedure for computing the surface sensitivities is used repeatedly in a gradient-based optimization framework in order to march the design surface shape toward an optimum through gradient projection and mesh deformation.

In SU^2 two methodologies for design variable definition are used. In two-dimensional airfoil calculations, Hicks-Henne bump functions are employed[27] which can be added to the original airfoil geometry to modify the shape. The Hicks-Henne function with maximum at point x_n is given by

$$f_n(x) = \sin^3(\pi x^{e_n}),\ e_n = \frac{\log(0.5)}{\log(x_n)},\ x \in [0,1], \tag{86}$$

so that the total deformation of the surface can be computed as $\Delta y = \sum_{n=1}^{N} \delta_n f_n(x)$, with N being the number of bump functions and δ_n the design variable step. These functions are applied separately to the upper and lower surfaces. After applying the bump functions to recover a new surface shape with each design iteration, a spring analogy method is used to deform the volume mesh around the airfoil.

In three dimensions, a Free-Form Deformation (FFD) strategy[78,80] has been adopted. Here an initial box encapsulating the object (rotor blade, wing, fuselage, etc.) to be redesigned is parameterized as a Bézier solid. A set of control points are defined on the surface of the box, the number of which depends on the order of the chosen Bernstein polynomials. The solid box is parameterized by the following expression

$$X(u,v,w) = \sum_{i=0}^{l}\sum_{j=0}^{m}\sum_{k=0}^{n} P_{i,j,k} B_i^l(u) B_j^m(v) B_k^n(w), \tag{87}$$

where l, m, n are the degrees of the FFD function, u, v, $w \in [0,1]$ are the parametric coordinates, $P_{i,j,k}$ are the coordinates of the control point (i,j,k), and $B_i^l(u)$, $B_j^m(v)$ and $B_k^n(w)$ are the Bernstein polynomials. The Cartesian coordinates of the points on the surface of the object are then transformed into parametric coordinates within the Bézier box. Control points of the box become design variables, as they control the shape of the solid, and thus the shape of the surface grid inside. The box enclosing the geometry is then deformed by modifying its control points, with all the points inside the box inheriting a smooth deformation. With FFD, arbitrary changes to the thickness, sweep, twist, etc. are possible for the design of any aerospace system. Once the deformation has been applied, the new Cartesian coordinates of the object of interest can be recovered by simply evaluating the mapping inherent in Eq. 87.

To increase the flexibility of the definition of three-dimensional design variables, a nested Free-Form Deformation (FFD) capability has been implemented. The key idea of this methodology is the use of a set of nested FFD boxes to explore the design space, with each FFD box corresponding to a different objective (see Fig. 9).

- The volume box (which embeds the entire aircraft surface) is useful for rotating the entire geometry, adjusting the area of the different sections or the total length of the aircraft.
- The main box (which embeds the main wing, fuselage, tail, etc.) is useful for redefining the camber and thickness of the wing, or applying some deformations like twist, sweep, etc.
- The secondary box (in small localized areas) is useful for removing shock waves in those areas (e.g., shocks induced by the nacelles).

H. Optimization framework

The adjoint formulation allows the computation of sensitivities of a wide range of objective functions commonly used for optimization problems: quadratic deviation from a target pressure (inverse design), drag minimization, lift maximization, pitching moment, aerodynamic efficiency, and linear combinations of those. Also, several constraints have been implemented: fixed non-dimensional flow parameters (minimum lift, maximum drag, etc.) and geometrical estimations (maximum and minimum thickness, curvature, volume, area, etc.).

With respect to the optimization strategy, two main options are available in SU^2: Gradient Based Optimization (GBO) and Surrogate Based Optimization (SBO). In this section both methods will be described.

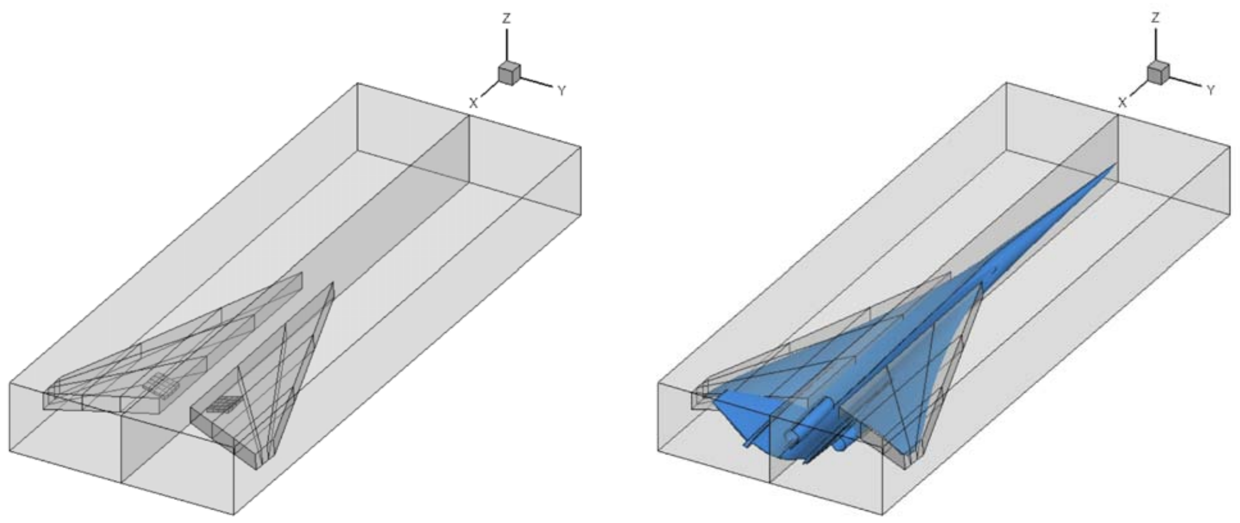


Figure 9. Nested Free-Form Deformation example (volume, main, and secondary boxes are shown).

1. *Gradient based optimization*

The gradient based optimization uses the SciPy library,[46] a well-established open-source software for mathematics, science and engineering. The SciPy library provides many user-friendly and efficient numerical routines for the solution of non-linear constrained optimization problems, such as conjugate gradient, Quasi-Newton or sequential least-squares programming algorithms. At each design iteration, the SciPy routines require as inputs the values and gradients of the objective functions as well as the chosen constraints, but not necessarily in a sequential order. Thus the shape optimization wrapper must also keep track of what simulations have been run to avoid unnecessarily repeating an analysis.

2. *Surrogate based optimization*

Surrogate-based optimization results in SU^2 were obtained using an in-house code based on Gaussian Process Regression (GPR) for response surface modeling.[55,56] Briefly, GPR is performed by conditioning a multivariate normal distribution,

$$f \sim \mathcal{N}\left(\mu, [\sigma]\right), \tag{88}$$

where f is a normally distributed function with mean vector μ and standard deviation matrix $[\sigma]$.

In SU^2, we take a uniformly zero mean vector and populate the standard deviation matrix with a covariance model, k, that is a function of training and estimated data:

$$\begin{bmatrix} f_p \\ f_k^* \end{bmatrix} \sim \mathcal{N}\left(0, \begin{bmatrix} k(x_p, x_q) & k(x_p, x_j^*) \\ k(x_k^*, x_q) & k(x_k^*, x_j^*) \end{bmatrix}\right), \\ \{\, f_i(x_i) \,|\, i = 1, ..., n \,\}\ ,\ \{\, f_t^*(x_t^*) \,|\, t = 1, ..., m \,\}. \tag{89}$$

The notation $(\cdot)^*$ is used to distinguish the estimated data from the training data. Additionally, index notation is used to describe the sub-blocks of the covariance matrix, where $k(x_p, x_q)$ would be equivalent to the matrix k_{pq}. There are n training point vectors, x, with function values, $f(x)$, and m estimated data point vectors, x^*, with function values, $f^*(x^*)$.

Of the data, we do not know the estimated function values f^*. We do know the training data locations x and function values $f(x)$, as well as the desired estimated data locations, x^*. Following Rasmussen's derivation,[74] we condition the normal distribution with the available data

$$f|x^*, x, f \sim \mathcal{N}\left(f^*, \mathbb{V}[f^*]\right), \tag{90}$$

which allows us to identify useful relations for estimating a function fit,

$$\begin{aligned} f_k^* &= k(x_k^*, x_q)\, k(x_p, x_q)^{-1}\, f_p, \\ \mathbb{V}[f_k^*] &= \left(k(x_k^*, x_j^*) \,-\, k(x_k^*, x_q)\, k(x_p, x_q)^{-1}\, k(x_p, x_j^*)\right)_k , \end{aligned} \tag{91}$$

where $\mathbb{V}[f^*]$ is the covariance of the estimated value f^*.

3. Covariance Function

The covariance function models the spatial correlation between data points. It is chosen based on the types of functions that are going to be modeled. Highly-smooth or weakly-smooth functions would be examples of different types of functions that would require different choices of covariance functions. A common covariance function is the Gaussian function of the Euclidean distance between points:

$$\begin{gathered} k(x_p, x_q) = k(p, q) = \theta_1^2 \exp\left(-\frac{1}{2\theta_2^2}\sum_{z=1}^{d}(p_z - q_z)^2\right), \\ \{p_i, q_i, \tfrac{\partial}{\partial x_i} \mid i = 1, ..., d\}, \end{gathered} \tag{92}$$

where d is the number of dimensions, and p and q are the position vectors chosen from the design space, x. There are two degrees of freedom in the covariance function known as hyper-parameters. In terms of their effect on the function fit, the nominal variance θ_1 is a measure of the amount of variance allowable between points, and the length scale θ_2 is a measure of the range of influence of a point.

We can adjust the definition of the kernel matrix to include the sensitivity information from an adjoint solution. In high-dimensional design spaces this could be a large amount of data. To implement this, we model the correlation between points and derivatives by taking the derivatives of the original covariance function.

Finally, adaptive refinement sampling criteria can be used to choose new locations in the design space to sample training data for improving the accuracy of the fit. These criteria take advantage of the probabilistic construction of GPR by constructing estimates of expected improvement, for example.

V. Capabilities

In this section, we present a number of the capabilities of SU^2 with a focus on high-fidelity analysis, optimization studies and grid adaptation.

A. High-fidelity analysis

Verification and Validation (V&V) is a critical requirement when building reliable CFD tools and, among other things, should address the consistency of the numerical methods, the accuracy for different important application cases, and sensitivity studies with respect to numerical and physical parameters. Here we apply SU^2 to several test cases and make appropriate comparisons to experiments in order to validate SU^2 suite of tools.

1. Compressible RANS simulations

The steady state Reynolds-averaged Navier Stokes (RANS) solver is a key component of the SU^2 suite. A wide range of general fluid dynamics analysis problems have been solved and compared to experimental results spanning subsonic, transonic, supersonic and hypersonic flight regimes, including:

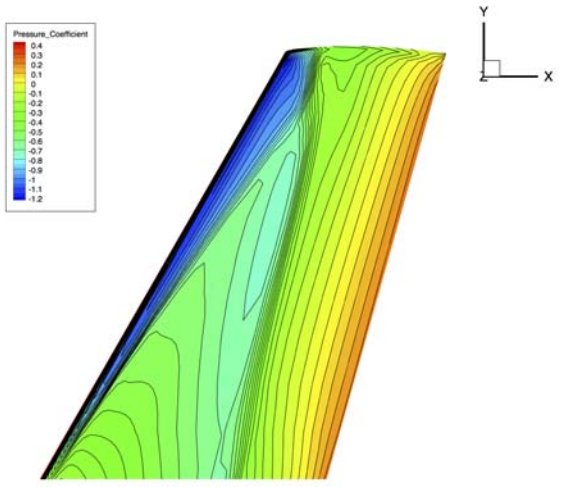


Figure 10. Pressure contours on the upper surface of the ONERA M6 wing (RANS-SA).

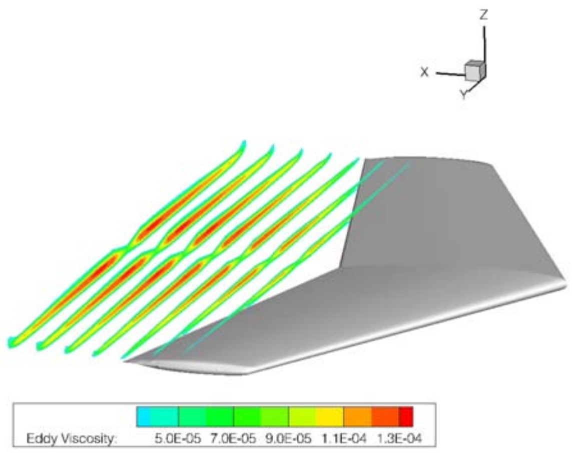


Figure 11. Eddy viscosity contours on the ONERA M6 wing (RANS-SA).

- ONERA M6 - A swept, semi-span wing with no twist that uses a symmetric airfoil (ONERA D sections). The aspect ratio is $\Lambda = 3.8$ and the leading edge angle is $\phi = 30.0°$. The simulation shown in Fig. 10 and Fig. 11 uses the flow field conditions of Test 2308: $M_\infty = 0.8395$, angle of attack 3.06°, angle of sideslip 0.0°. These correspond to a Reynolds number of 11.72 million based on the mean aerodynamic chord, $c = 0.64607$ m. The ONERA M6 was tested in a wind tunnel at transonic Mach numbers (0.7, 0.84, 0.88, 0.92) and various angles of attack up to 6°. The wind tunnel tests are documented by Schmitt and Charpin.[79] In Fig. 12 the comparison with experimental data is shown.

- RAE 2822 - 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 AGARD Report by Cook et al.[15] . The Reynolds number is 6.5 million based on a unit chord length, $M_\infty = 0.729$, and the airfoil is inclined at an angle of attack of 2.31°. Figs. 13 and 14 show the pressure contours compared with experimental data.[15]

- Zero-pressure flat plate - For the verification and validation of turbulence models a standard test case is the turbulent flow over a flat plate. The flow is everywhere turbulent and a boundary layer

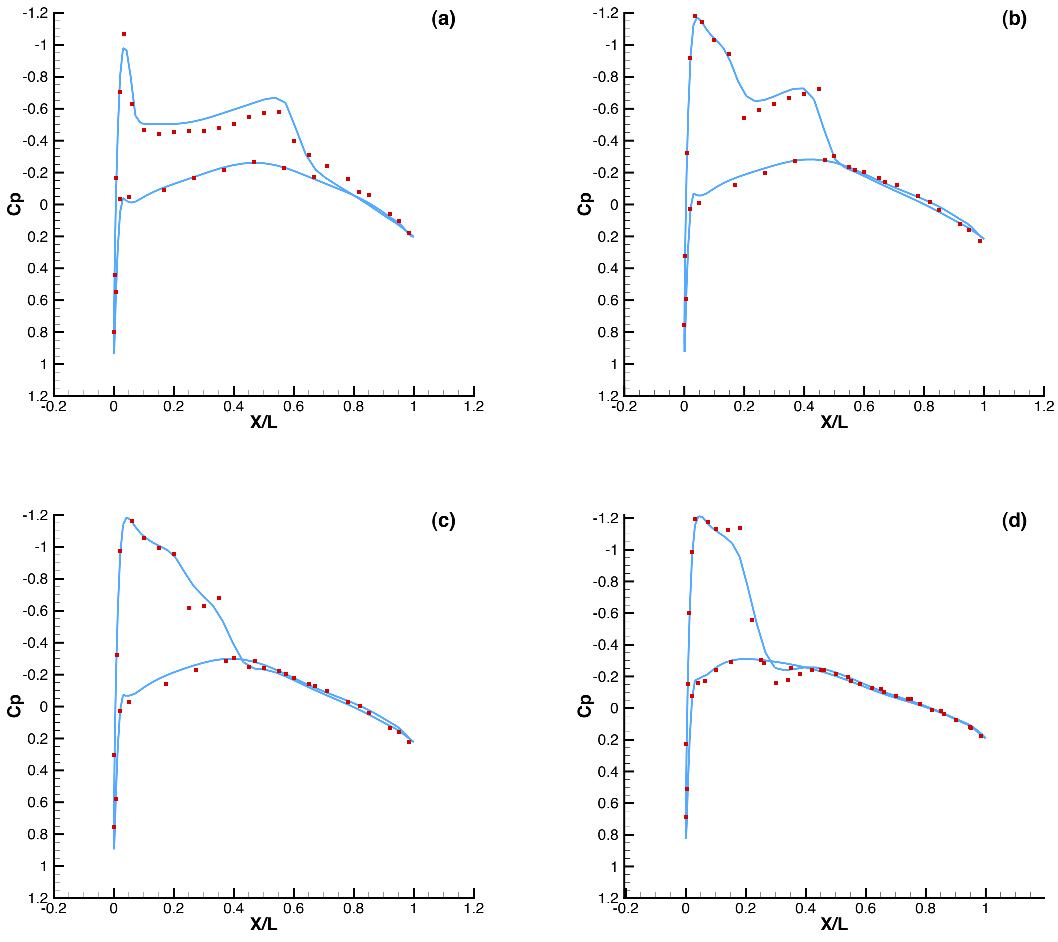


Figure 12. Comparison of C_p profiles of the experimental results of Schmitt and Carpin (red squares) against SU^2 computational results (blue line) at different sections along the span of the wing. (a) y/b = 0.2, (b) y/b = 0.65, (c) y/b = 0.8, (d) y/b = 0.95.

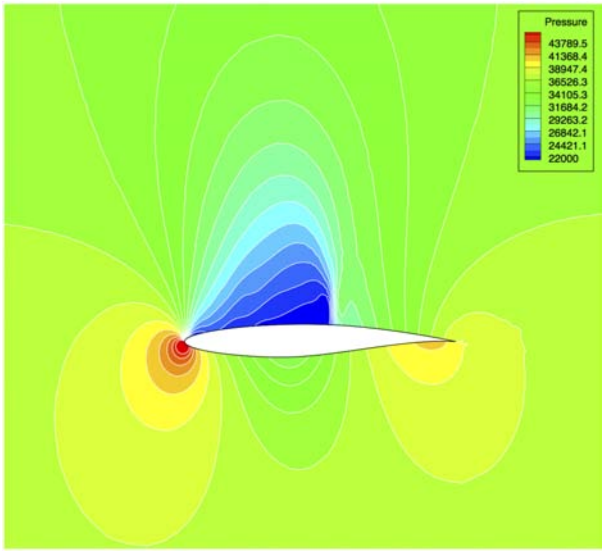


Figure 13. Pressure contours (Pa) on the RAE 2822 airfoil (RANS-SA).

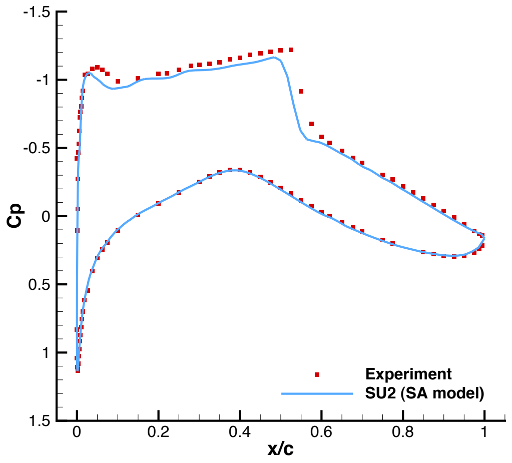


Figure 14. Pressure coefficient comparison with experimental data on the RAE 2822 airfoil (RANS-SA).

develops over the surface. The lack of separation bubbles or other more complex flow phenomena allows turbulence models to predict the flow with a high level of accuracy. The length of the flat plate used here is 2 m, and it is represented by an adiabatic no-slip wall boundary condition. The S-A turbulence variable is plotted in Fig. 15. The Reynolds number based on a length of 1 m is 5 million, and the $M_\infty = 0.2$. For validation purposes the u^+ vs. y^+ profiles are compared against theoretical profiles of the viscous sublayer and log law region in Fig. 16.

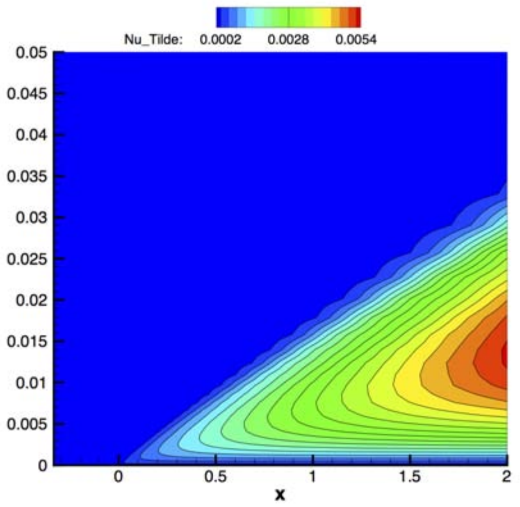


Figure 15. Turbulence variable on a subsonic flat plate (RANS-SA).

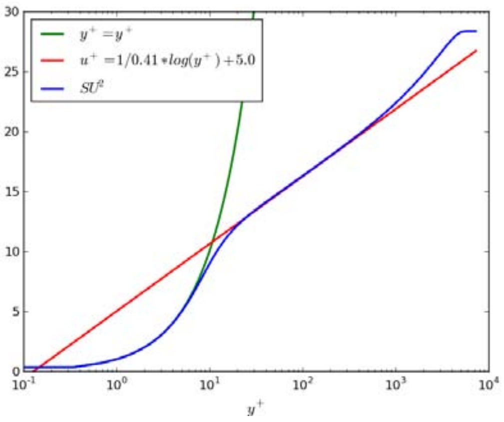


Figure 16. Velocity profile comparison against the law of the wall on a subsonic flat plate(RANS-SA).

2. *Rotating frame simulations*

A rotating frame formulation of the RANS equations is included in SU^2 for efficient, steady analysis of fluid around rotating aerodynamic bodies. Potential applications include wind turbines, turbomachinery, propellers, open-rotors, or helicopter rotors. The Caradonna and Tung[11] rotor blade was modeled in inviscid flow to validate the rotating frame formulation as compared to experiment. The rotor geometry consists of two untwisted, untapered blades with an aspect ratio of 6 and a constant NACA 0012 airfoil section along the entire span. For comparison purposes, a lifting case was chosen with a collective pitch angle of 8° and a pre-cone angle of 0.5°. The flow conditions are that of hover at 1250 RPM which results in a tip Mach number of 0.439. Fig. 17 contains the C_p contours on the upper blade surface along with C_p distributions at several radial stations compared to experiment.

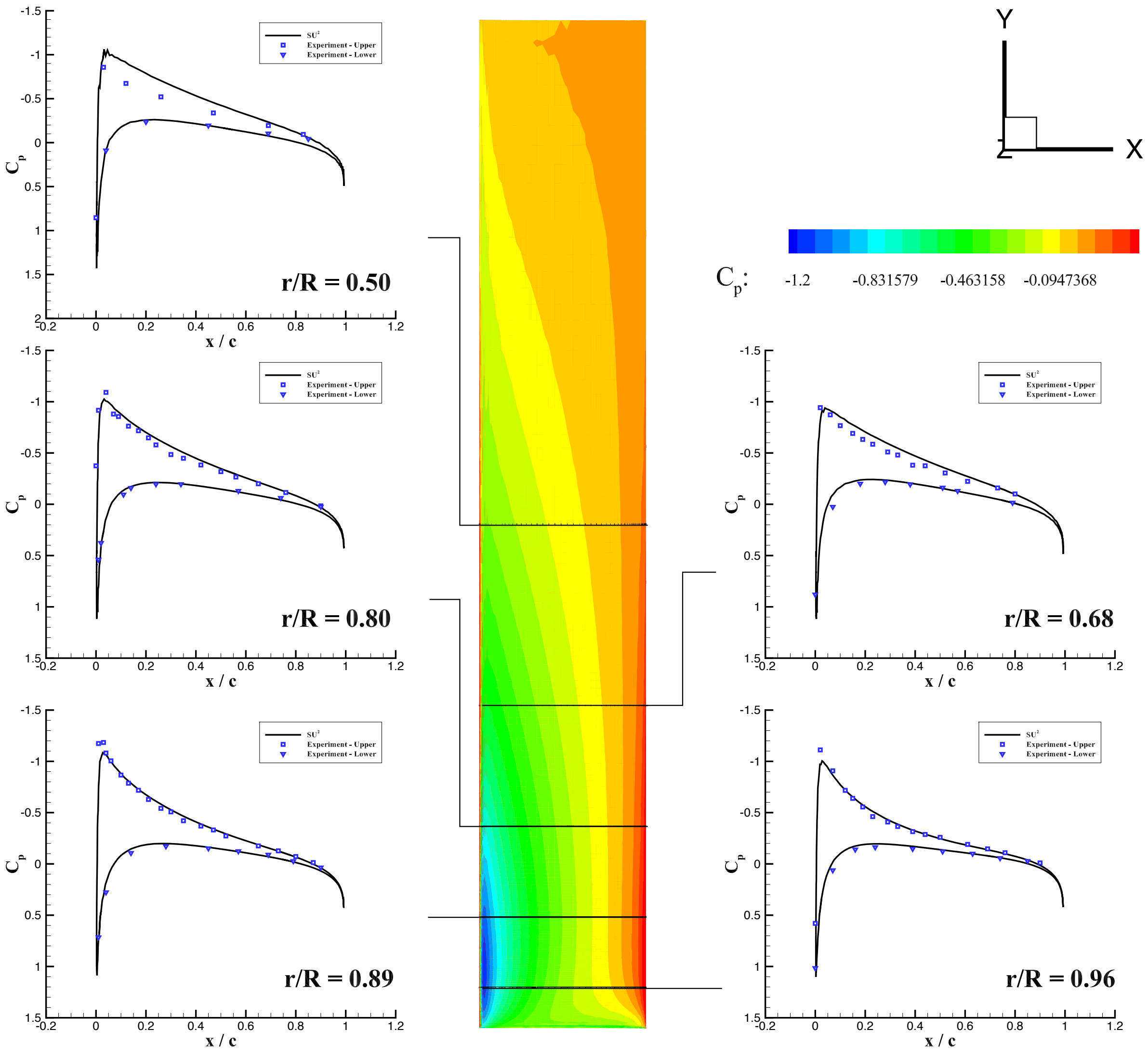


Figure 17. Contours over a full revolution, upper surface C_p.

3. Supersonic simulations

SU^2 is able to compute the near-field pressure signature as well as the equivalent area,[1,67] features that are important for the simulation of supersonic aircraft. This has been applied to the Lockheed N+2 aircraft baseline geometry shown in Fig. 18, at $M_\infty = 1.7$ and an angle of attack of 2.1°.

A grid refinement study has been performed to reduce the computational expense. More specifically, we coarsened the resulting mesh from 3.1 million nodes to 1.2 million nodes. Fig. 19 shows the similarities in

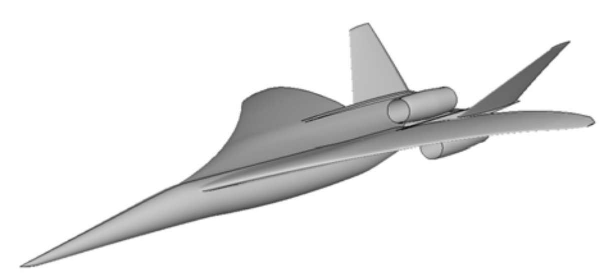


Figure 18. Baseline LMCO N+2 supersonic passenger jet geometry.

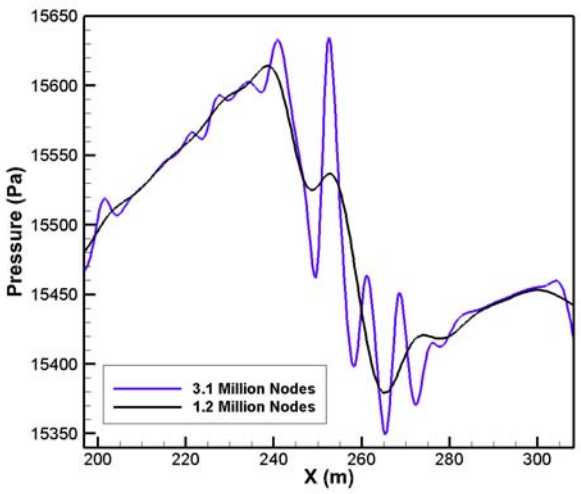


Figure 19. Comparison of near-field pressure signatures for coarse and fine mesh.

the near-field pressure signature between the two meshes, while a comparison of the pressure field on the symmetry plane in Fig. 20 and Fig. 21 also shows similar behavior.

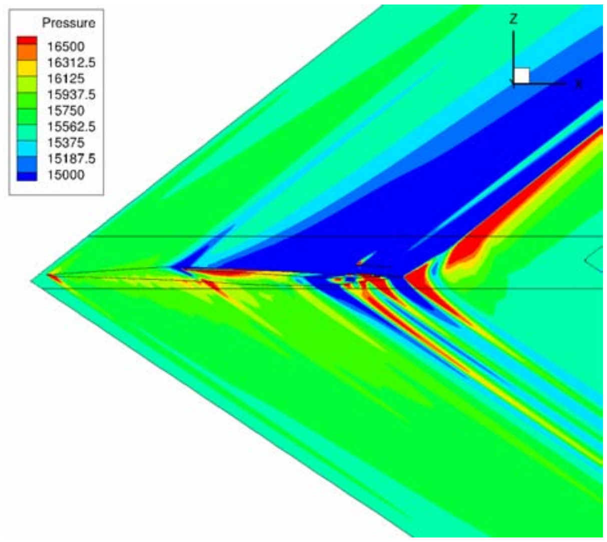


Figure 20. Pressure contours for fine mesh.

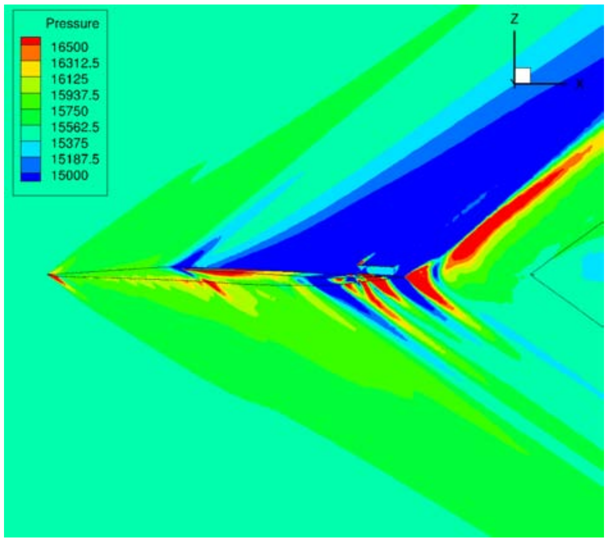


Figure 21. Pressure contours for coarse mesh.

4. Low-Mach number simulations

A NACA0012 airfoil at 2° angle of attack at $M_\infty = 0.01$ was used to compare results of the standard Roe scheme and the Roe-Turkel scheme at low Mach numbers. The simulation is inviscid, second-order accurate in space, and uses implicit time integration with three levels of multi-grid for convergence acceleration. The solution with Roe-Turkel converged by six orders of magnitude while the solution using Roe's scheme did not converge. Fig. 22 shows the distribution of non-dimensional pressure over the airfoil from the two methods. The coefficient of lift over the NACA 0012 airfoil at $M_\infty = 0.01$ and 2° angle of attack is 0.235.

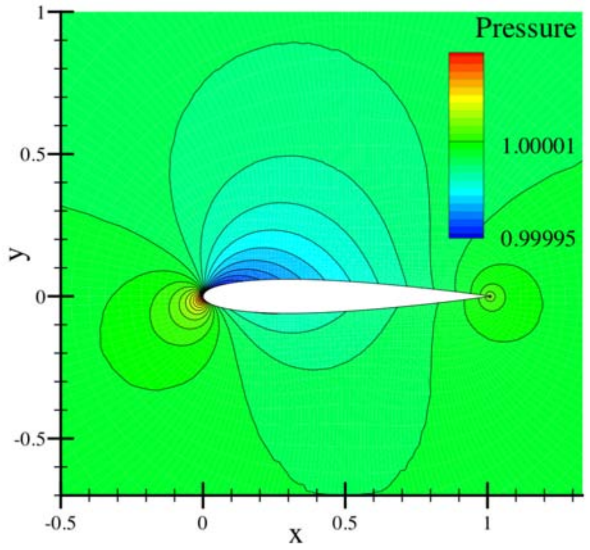


Figure 22. Non-dimensional pressure over NACA 0012 at $M_\infty = 0.01$ using the Roe-Turkel scheme.

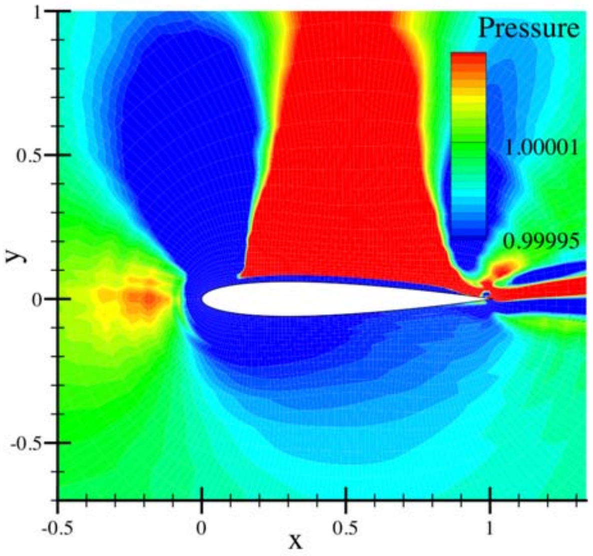


Figure 23. Non-dimensional pressure over NACA 0012 at $M_\infty = 0.01$ using Roe's scheme (non-converged solution).

5. *Time-accurate simulations with dynamic meshes*

Several cases have been used for testing the implementation of the unsteady governing flow equations in Arbitrary Lagrangian-Eulerian (ALE) form.

- Pitching NACA 64A010 airfoil - a comparison was made against the well-known CT6 data set of Davis[18] for validation purposes. The physical experiment measured the unsteady performance of the NACA 64A010 airfoil pitching about the quarter-chord point. The particular experimental case of interest studied pitching motion with a reduced frequency, w_r, of 0.202 ($\omega_r = \frac{\omega \cdot c}{2 v_\infty}$, where c is the chord and ω is the angular frequency of the pitching), $M_\infty = 0.796$, a mean angle of attack of 0°, and a maximum pitch angle of 1.01°.

  The numerical simulations were performed with 25 time steps per period for a total of 10 periods. Fig. 24, and Fig. 25 show a comparison of the lift coefficient versus angle of attack between SU^2 and experiment during the final period of oscillation for the Euler and RANS equations, respectively. In physical time, the curve is traversed in a counterclockwise fashion. Note that non-linear behavior corresponding to moving shock waves results in a hysteresis effect.

- UH-60A rotor blade - A single blade was modeled in inviscid flow to demonstrate the capability of this technique.[14] To minimize wake interactions between the blades, a high-speed forward flight case (C8534) was chosen. The blade surface mesh is shown in Fig. 26.

  The CFD high-fidelity results are coupled with a comprehensive structural dynamics (CSD) tool, the University of Maryland Advanced Rotorcraft Code (UMARC). UMARC solves the coupled structural dynamic and control problem to yield a trimmed vehicle (including rotor and body). Its structural simulation depends upon an external aerodynamic analysis to provide the performance of each airfoil section along the radius of the blade at each azimuth.

  Mesh deformations (displacements and rotations of the airfoil sections along the blade) from UMARC were mapped to the three-dimensional surface mesh at each time (azimuth) step. Three full revolutions were taken to overcome any initialization effects and achieve a periodic steady state. The aerodynamic properties of interest (in this case C_Q and C_T) are typically averaged over all time instances of a periodically-steady revolution. The time histories of C_Q and C_T are shown in Fig. 27. As expected for this flight case, periodic flow emerged rapidly with the second and third revolutions of the flow

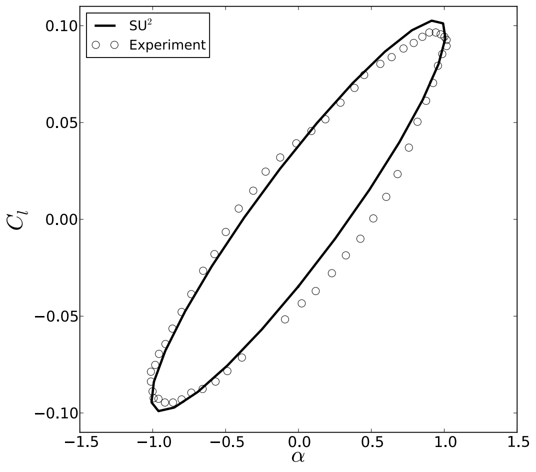


Figure 24. Coefficient of lift hysteresis comparison against experimental data (Euler simulation).

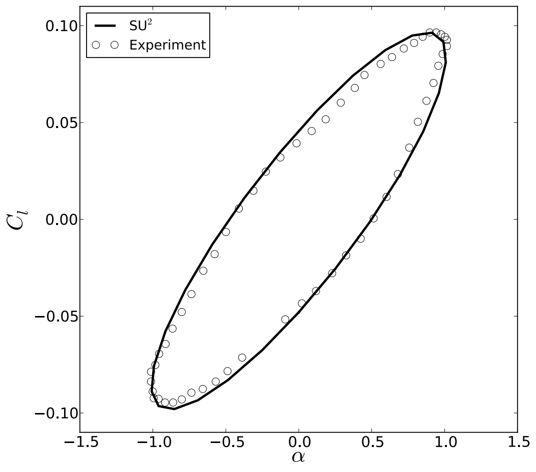


Figure 25. Coefficient of lift hysteresis comparison against experimental data (RANS simulation).

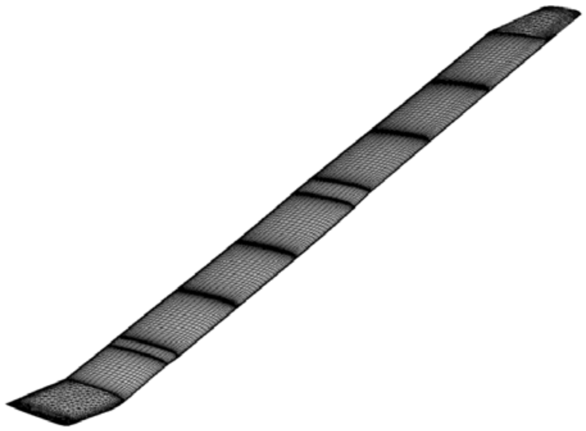


Figure 26. UH-60A rotor blade surface mesh.

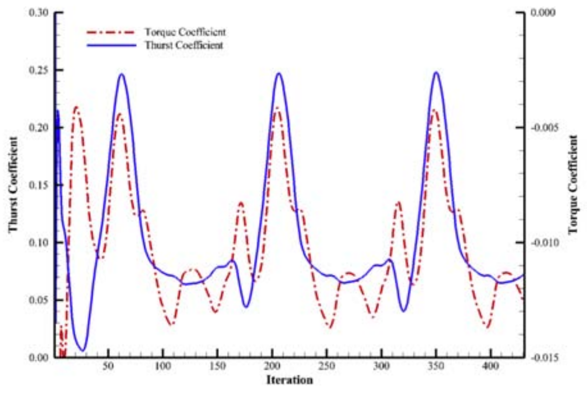


Figure 27. Time histories of C_Q and C_T for three revolutions of a flow solution.

solution nearly exactly equal. Contours of the flow solutions for a full revolution are shown in Fig. 28 and Fig. 29.

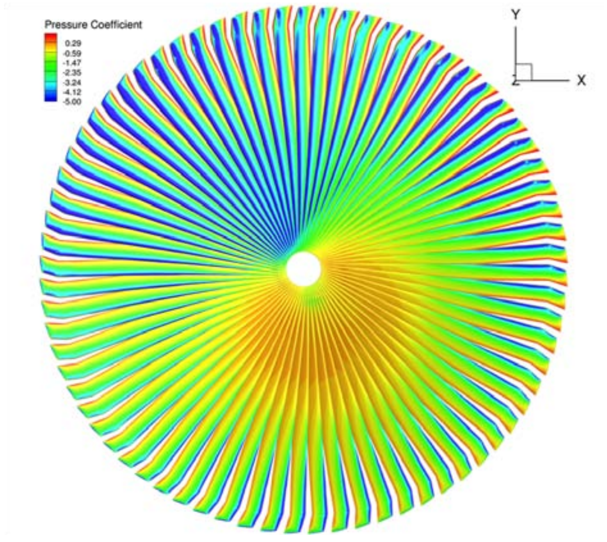


Figure 28. Pressue contours over a full revolution, upper surface C_p.

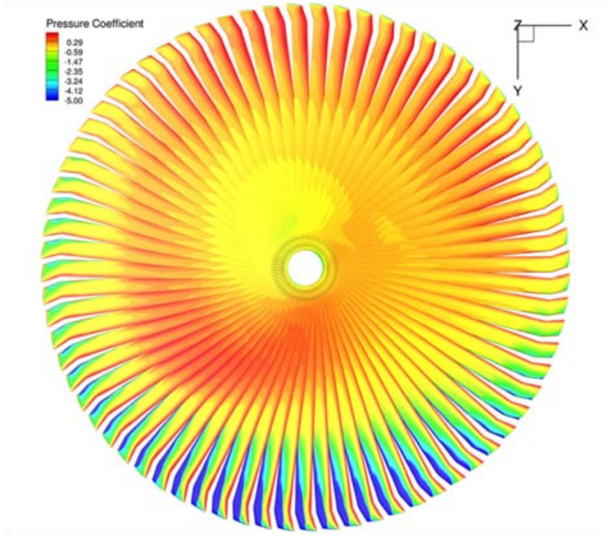


Figure 29. Pressure contours over a full revolution, lower surface C_p.

6. *Free-surface simulations*

A 2-D, super-critical and sub-critical steady-state flow over a submerged bump without breaking waves is presented as a baseline test of the free-surface solver verification and validation. The selected 2-D bump[68] has the following shape: $z = (2.7/4)x(x-1)^2, \quad 0 \le x \le 1$, and is placed on the bottom of a channel. For the cases studied here, a final steady-state solution is achieved. Two free-surface cases with different Froude numbers (based on the bump length, $L = 1.0$) and undisturbed depths of water, H, were selected: a super-critical case, $Fr_L = 1.0$ and $H = 0.228$, and a sub-critical case, $Fr_L = 0.304$ and $H = 0.500$.

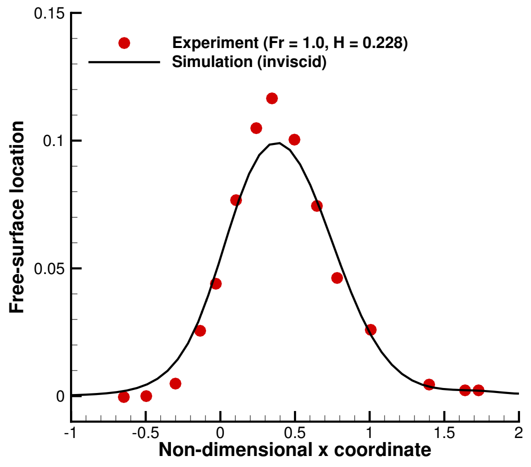


Figure 30. Comparison between experiments and inviscid simulation (super-critical case,).

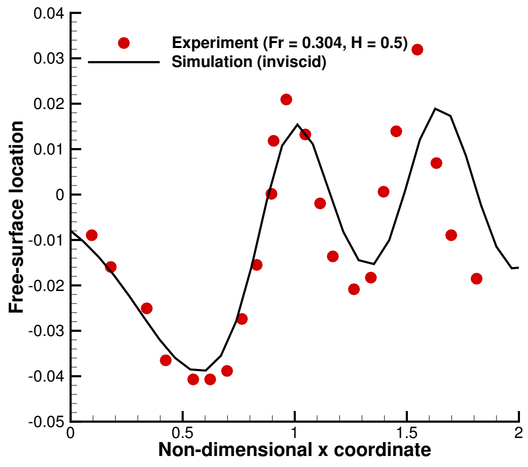


Figure 31. Comparison between experiments and inviscid simulation (sub-critical case).

Both the super-critical and sub-critical simulations admit steady-state solutions and have been inves-

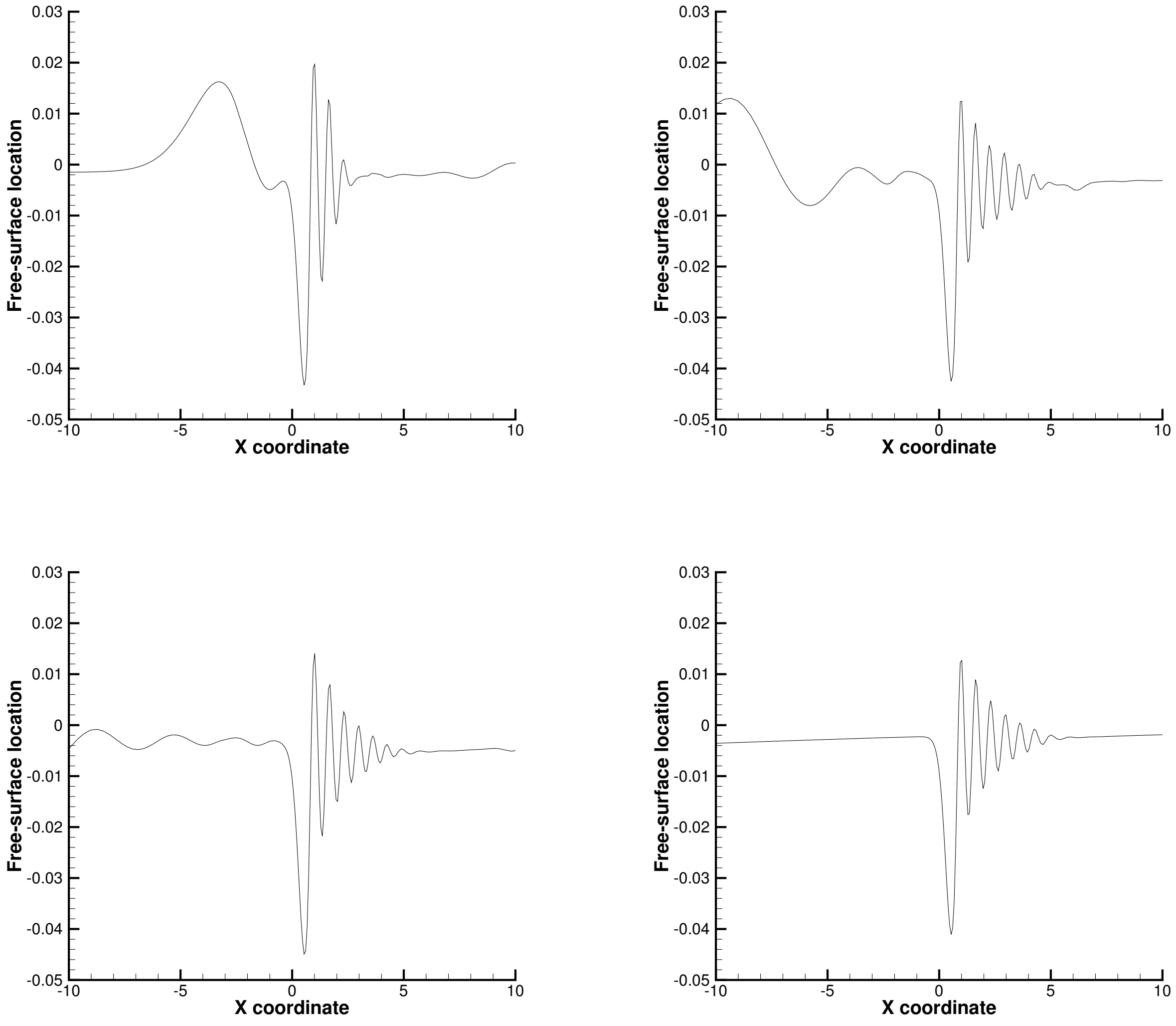


Figure 32. Sub-critical case, Cahouet experiment. Iterations: (top left) 10, (top right) 20, (bottom left) 30 and (bottom right) 590 (steady state).

Quantity	Argon	Ions	Electrons
Temperature (K)	810	810	3900
Density (kg/m^3)	1.664 $\times 10^{-4}$	1.664 $\times 10^{-6}$	2.29 $\times 10^{-11}$
Velocity (m/s)	(2347, 0, 0)	(2347, 0, 0)	(2347, 0, 0)
Mach number	4.6	4.6	0.0077

Table 4. Flow initial conditions

tigated by Cahouet.[9] In Fig. 30, a comparison between the simulation and super-critial experiment is presented. In Fig. 31, the sub-critical case is compared with the experiments. In both cases the wave profile shows good agreement with measurements, and we note that the under-predicted second wave crest has been reported by other numerical experiments. In Fig. 32 the time evolution of the simulation is presented.

7. *Plasma simulation*

SU^2 can model ionized gases, and to validate this part of the solver, we simulated the flow of partially ionized Argon gas over a solid body and computed the heat transfer to a three-dimensional surface from the plasma stream. The results for the computed heat flux at the wall are then compared with experiments performed by R. J. Nowak et al.,[65] in which Argon gas was passed over a d.c. arc heater which ionized it to 1%. This stream of ionized Argon was expanded to Mach 4.6 through a nozzle at the end of which measurements of flow parameters such as the electron temperature, density and others were made.

A hybrid, unstructured, pre-adapted, three-dimensional mesh was used for this simulation (see Fig. 34), and the flow for the three species was started with values corresponding to the inlet conditions given in Table 4.

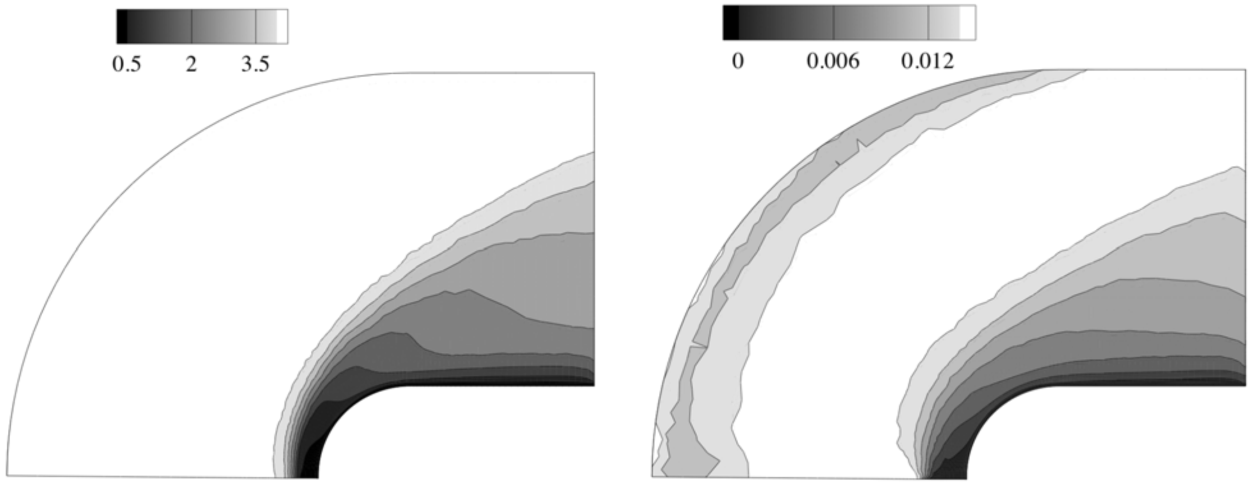


Figure 33. Distribution of Mach number of: (left) argon neutral atoms, (right) electrons. Note that the neutral atoms were supersonic, while the electrons were subsonic. This difference is due to the speed of sound in electrons, which is large owing to their small mass.

Catalytic boundary conditions were used on the wall of the shell, and heat released during recombination was transferred to the wall through an isothermal boundary condition for ions and neutrals and an adiabatic boundary condition for electrons. Heat flux from the Argon gas and ions to the shell was added to obtain the total heat flux to the shell (electrons did not transfer heat to the wall owing to the specific wall boundary condition).

Fig. 35 shows the numerically computed heat flux along the nose which matches very well with experiment. Fig. 33 shows the distribution of the Mach number for two species, neutral argons atoms and electrons. The velocity and temperature of the species were similar owing to inter-species collisions, but their Mach numbers vary significantly due to different physical properties. The very small mass of electrons compared to the

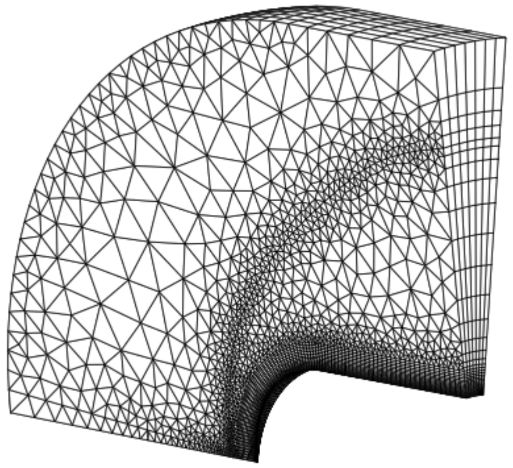


Figure 34. A tetrahedral-hexahedral hybrid mesh of 46,000 cells for simulation of a plasma stream over a sphere-cylinder body. This mesh is pre-adapted to capture the shock wave and the boundary layer.

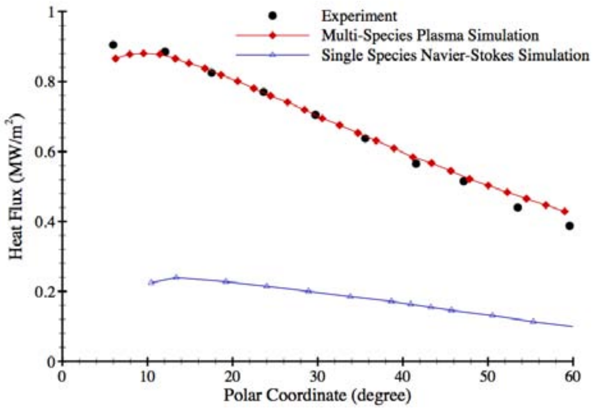


Figure 35. Comparison of numerical heat flux to the three dimensional shell from a supersonic stream of plasma in Argon with results from NASA experiments.

other chemical species results in a much larger speed of sound in electrons making their Mach numbers extremely small.

B. Optimal shape design and optimization

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 and Mesh Deformation Code, delivers objective function gradient information to optimization algorithms which enables surface shape optimization for complex geometries.

1. Supersonic aircraft design

The objective of this particular problem is to demonstrate the ability of the adjoint formulation to enable design optimization while maintaining a target equivalent area distribution across multiple azimuths.[55] More specifically, a gradient-based, multi-objective optimization of the Lockheed N+2 design was performed ($M_\infty = 1.7$, angle of attack $2.1°$). This was done for 9 iterations with azimuths varying from $0°$ to $60°$ (pressure disturbances above $60°$ do not reach the ground) and the drag was reduced from 0.00875 to 0.00850, a 3% reduction.

The ability to compute sensitivities is a fundamental capability and very useful output of SU^2. Fig. 38 and Fig. 39 show the sensitivities of C_D and C_L with respect to variations of the geometry in the local normal direction. The magnitude of the surface sensitivity is related to changes in the cost function caused by changes in geometry, and designers can use this sensitivity information to determine appropriate parameterizations of the configuration prior to optimization.

Fig. 36 and 37 show a comparison of the near-field pressure distributions, and Fig. 40 and Fig. 41 show a comparison of equivalent areas between the baseline and final design. The plots for equivalent area distribution show at most a 2% change from the target.

The overlay of the baseline and final geometry in Fig. 42 shows that the upper wing surface was flattened and the lower fuselage deformed inwards in order to reduce the drag. To compensate for the fuselage's effect on the equivalent area, the lower wing surface was deformed downwards as well. A comparison of surface pressures for the design study are given in Fig. 43, and contours of the density and adjoint density fields are shown in Fig. 44 and Fig. 45.

An unconstrained, surrogate-based drag optimization was performed on the N+2 configuration using the 9 Free-Form Deformation control points placed on the upper wing surface (see Fig. 46). A comparison of the convergence history with a gradient-based optimization of the problem is shown in Fig. 47. The

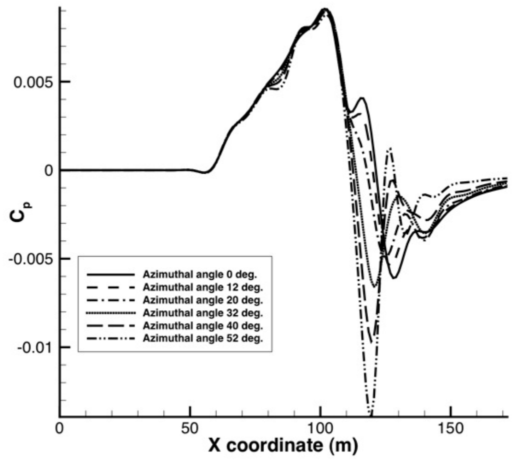


Figure 36. Baseline near-field pressure signatures.

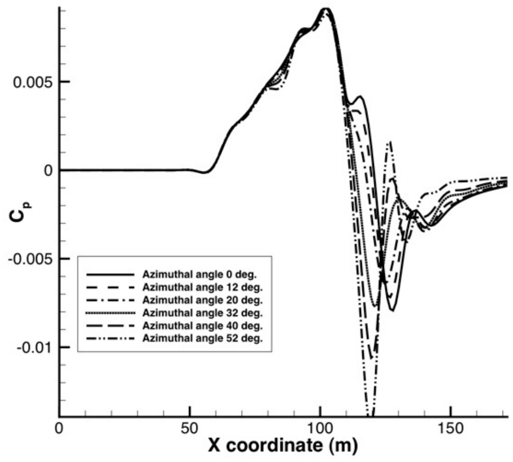
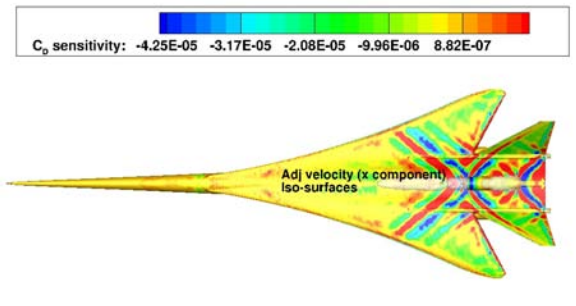
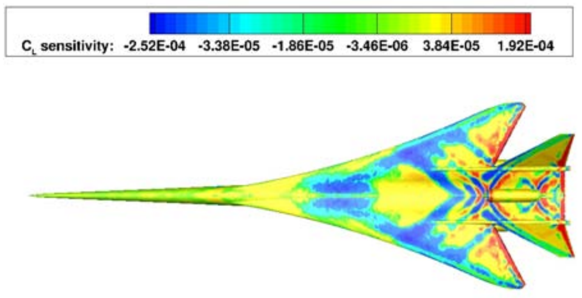


Figure 37. Final near-field pressure signatures.



C_D sensitivity: -4.25E-05 -3.17E-05 -2.08E-05 -9.96E-06 8.82E-07
Adj velocity (x component) Iso-surfaces

Figure 38. Drag sensitivity ($M_\infty = 1.7$, AoA $2.1°$), including propulsion effects.



C_L sensitivity: -2.52E-04 -3.38E-05 -1.86E-05 -3.46E-06 3.84E-05 1.92E-04

Figure 39. Lift sensitivity ($M_\infty = 1.7$, AoA $2.1°$), including propulsion effects.

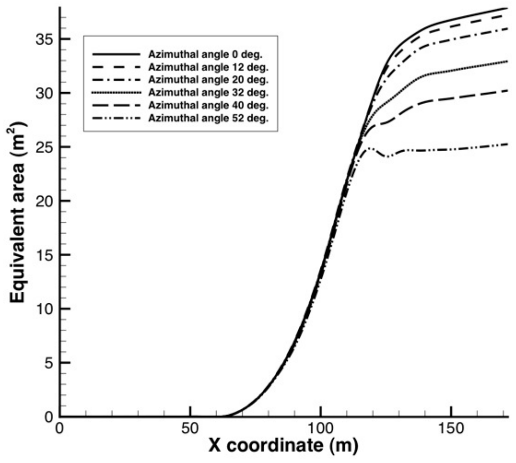


Figure 40. Baseline target equivalent area distributions.

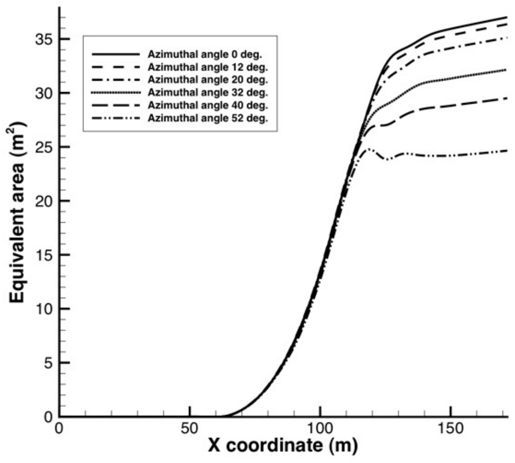


Figure 41. Final target equivalent area distributions.

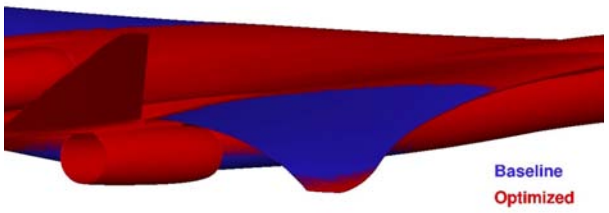


Figure 42. Baseline and final N+2 geometry comparison.

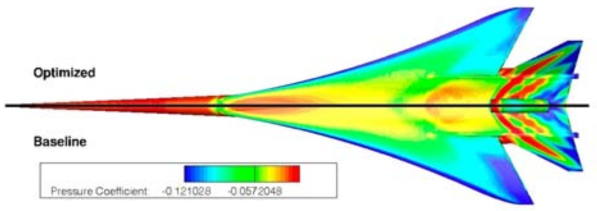


Figure 43. Baseline and final N+2 surface pressures.

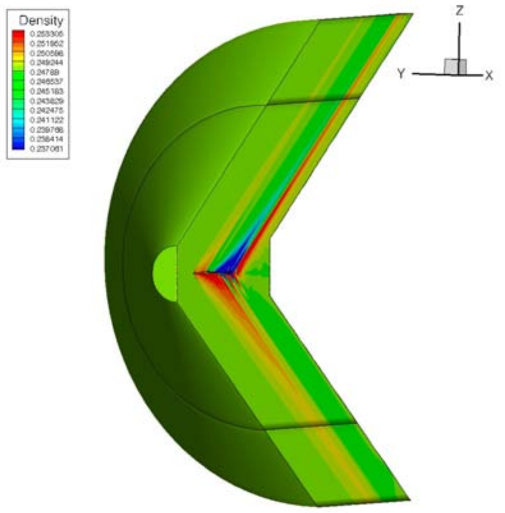


Figure 44. Contour plot of density for the N+2 baseline configuration.

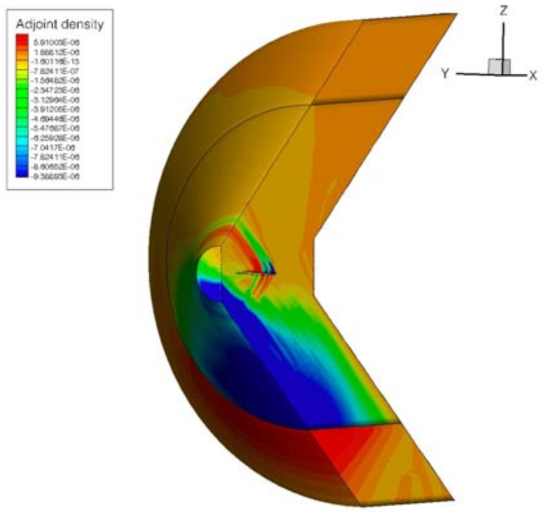


Figure 45. Contour plots of density adjoint variable for the N+2 baseline configuration and equivalent area as objective function.

surrogate-based optimization procedure was able to discover a 4.6% reduction in drag in 20 fewer iterations than a gradient-based optimization. The first 21 design points are chosen by latin hypercube sampling and are independent. They can be simulated simultaneously given enough computing resources, which further reduces the wall time spent optimizing. The six adaptive refinement iterations after the initial sample show that the response surface allows rapid convergence to the minimum. The use of an expected improvement sampling criteria also allows us to claim with reasonable certainty that this is the global minimum within the box bounds of the problem.

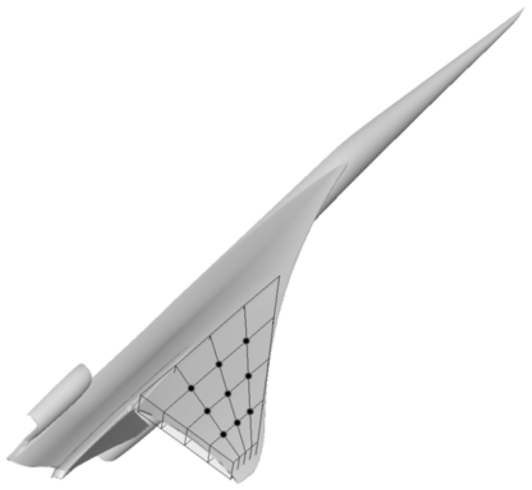


Figure 46. Location of the design variable (control points of the FFD box).

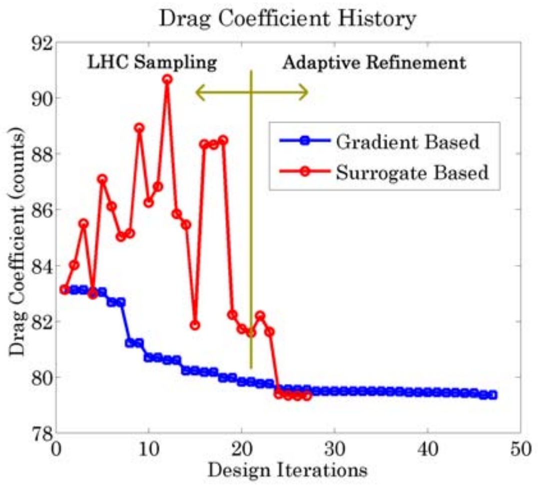


Figure 47. Comparison of unconstrained drag GBO and SBO.

A comparison of the geometry change between the baseline and final design in Fig. 48 shows that the drag reduction was accomplished by reducing the thickness of the wing.

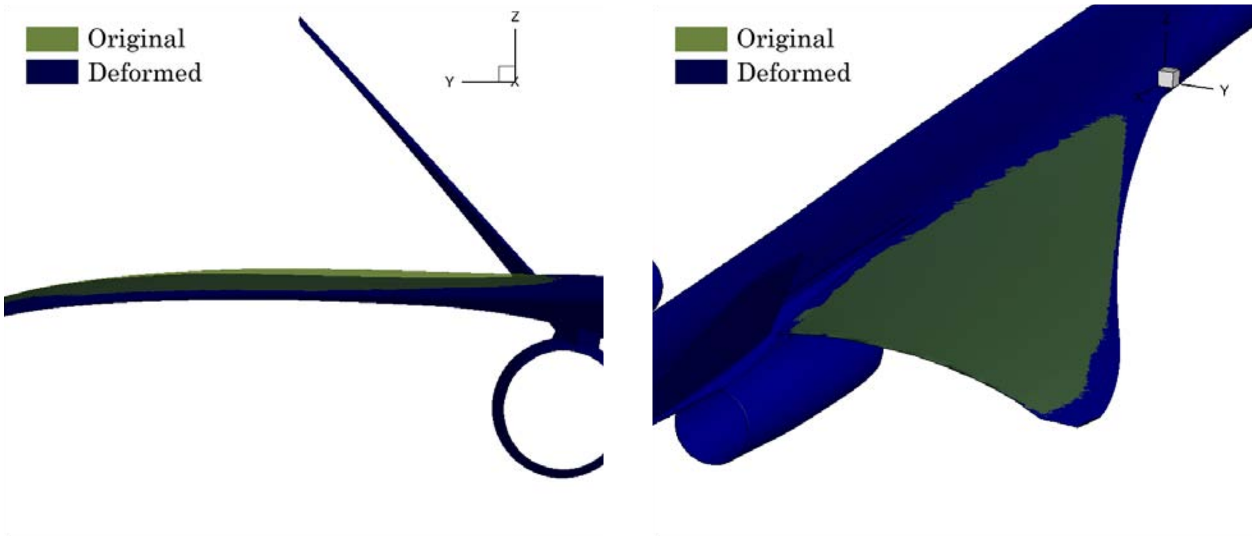


Figure 48. Original and deformed N+2 surfaces, unconstrained drag SBO.

2. *Airfoil and fixed wing optimization*

- NACA 0012 - A redesign of the NACA 0012 was undertaken, minimizing the drag while maintaining the lift and pitching moment of the original configuration. The simulation was performed at $M_\infty = 0.75$ and an angle of attack of 1.25°. Fig. 49 shows the initial and the final pressure distribution and Fig. 50 details the optimization evolution. After 12 iterations, the drag was reduced 200 counts, while the lift and pitching moment both remain constant.

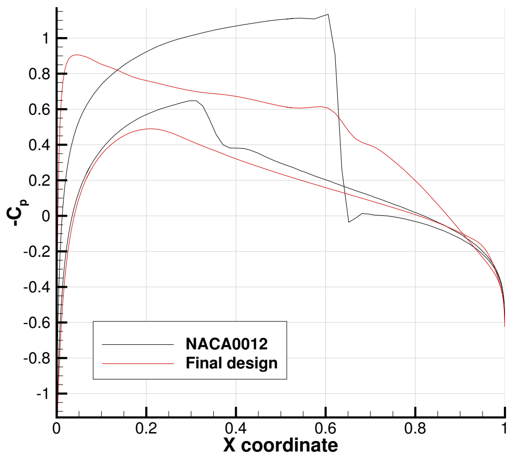


Figure 49. Pressure distribution over a NACA 0012 (baseline and final design).

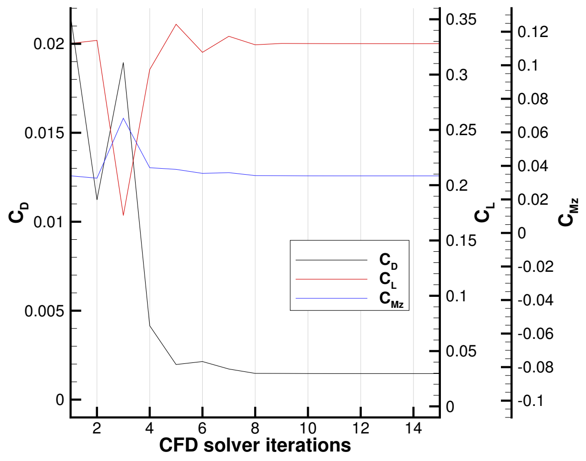


Figure 50. Transonic NACA 0012 optimization history (drag, lift, and pitching moment).

- ONERA M6 and DLR F6 - Both the ONERA M6 fixed wing ($M_\infty = 0.8395$, angle of attack 3.06°) and DLR F6 aircraft ($M_\infty = 0.75$, angle of attack 0°) were used as starting points for optimization, considering inviscid, transonic flow. The design variables were defined using the Free-Form Deformation (FFD) methodology, with the goal of the design process to minimize the coefficient of drag by changing the shape without any constraints. The specific design variables used were the z-coordinates of certain control points.

  In Figs. 51 and 52, it can be seen that the shock on the upper surface of the wing was removed, noting the change in shape of both the wing surface and the FFD box. Figs. 53 and 54 show the same technique applied to the aircraft configuration, noting that this time to shock is weakened but not entirely removed.

3. *Wing design using RANS equations*

A single-point minimization case is used to study the continuous adjoint method for turbulent flows.[8] The selected flow conditions were $M_\infty = 0.8395$, angle of attack = 3.06°, and Reynolds number of 11.72 million based on the mean aerodynamic chord $c = 0.64607\,\text{m}$. The profile being designed was an ONERA M6 wing. Only the upper surface of the wing was redesigned using the control points in Fig. 55.

After 10 iterations, the drag coefficient is finally reduced by a total of 10%. Fig. 56 shows the pressure coefficient contours of the original ONERA M6 wing compared to the redesigned profile. The shock wave initially present on the upper surface has been diminished.

4. *Redesign of a rotor in hover*

The Caradonna and Tung rotor geometry was used as an example of rotor design.[20] A lifting case was chosen with a collective pitch angle of 8°, a pre-cone angle of 0.5°. The flow conditions were that of hover at 2500 RPM results in a tip Mach number of 0.877.

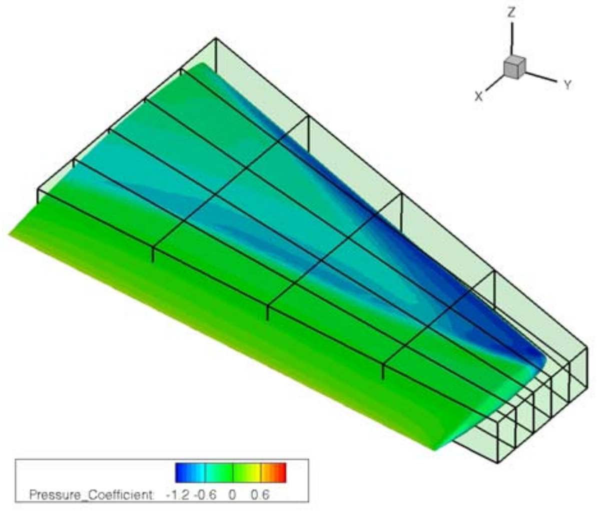


Figure 51. Pressure contours showing transonic shocks on the initial design (ONERA M6).

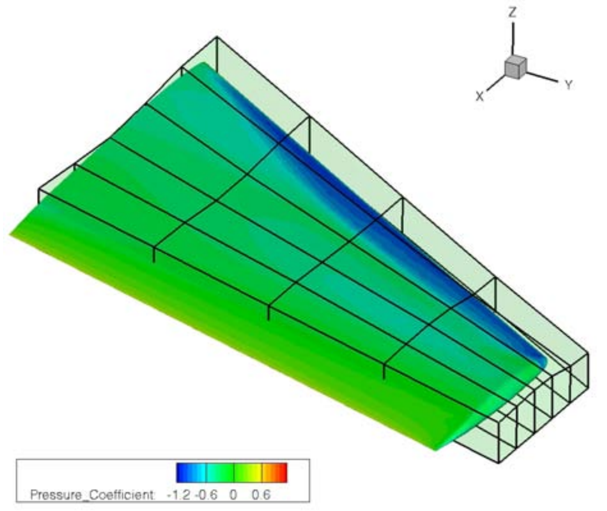


Figure 52. Pressure contours around the final wing design (ONERA M6).

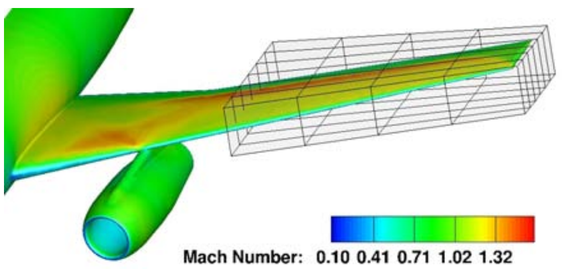


Figure 53. Pressure contours showing transonic shocks on the initial design (DLR F6).

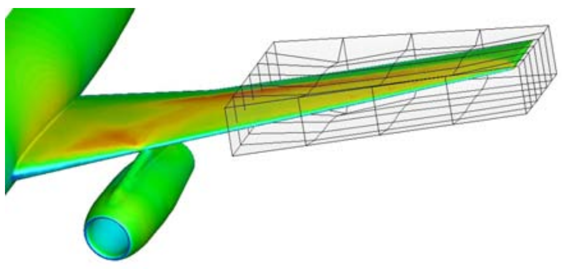


Figure 54. Pressure contours around the final aircraft design (DLR F6).

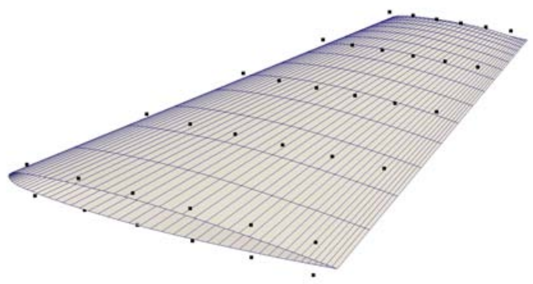


Figure 55. Control points of the Free Form Deformation box around the ONERA M6 wing..

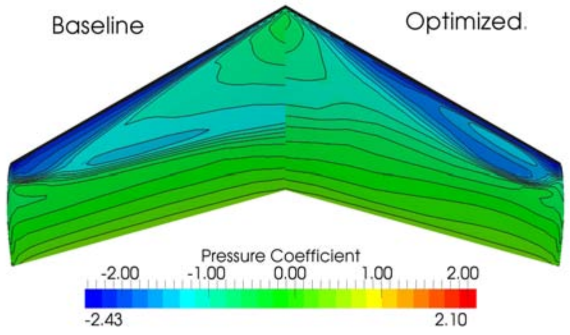


Figure 56. Coefficent of pressure contours on the initial and optimized ONERA-M6 wing.

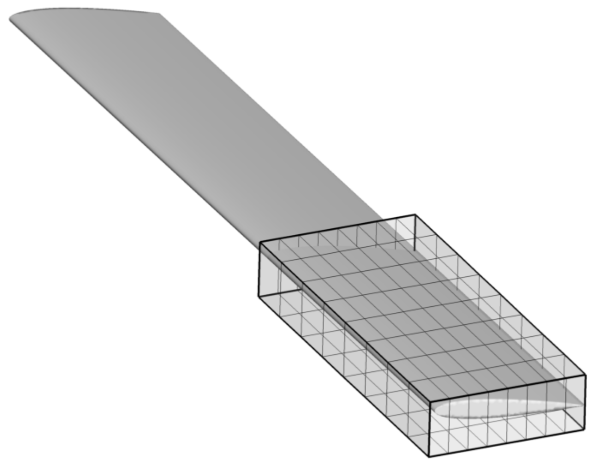


Figure 57. Rotor geometry with the FFD box surrounding the blade tip.

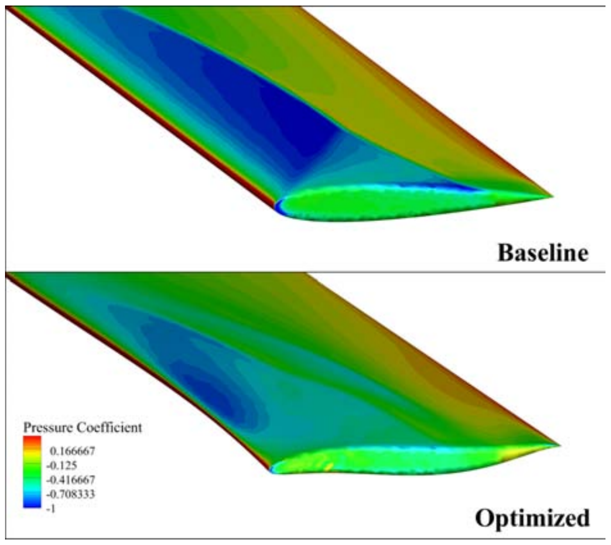


Figure 58. Comparison of the baseline and optimized rotor geometries with C_p contours.

Design variables were defined using a FFD parameterization. Movements in the vertical direction were allowed for a total of 84 control points on the upper and lower surfaces of the FFD box as shown in Fig. 57. In order to maintain a smooth surface during deformation, control points near the trailing edge and inboard side of the box were held fixed. The gradients of torque coefficient, C_Q, with respect to a subset of the FFD control point variables on the upper surface given by both the continuous adjoint and finite differencing were compared with results shown in Fig. 59.

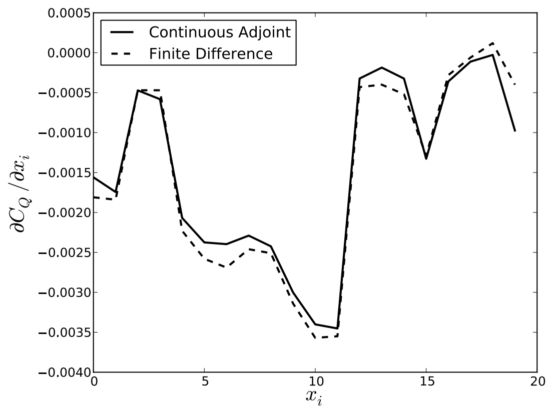


Figure 59. Continuous adjoint and finite differencing gradient comparison for 19 FFD control point variables.

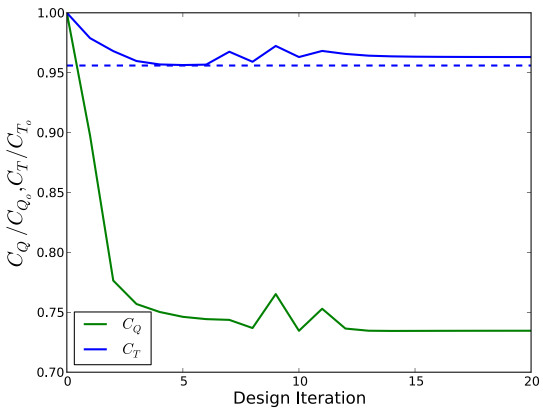


Figure 60. Gradient verification using the FFD control point variables and optimization results.

A redesign of the rotor blade shape for minimizing torque with a minimum thrust constraint of $C_T = 0.0055$ was performed using gradient information obtained via the continuous adjoint approach. After 20 design cycles, C_Q was reduced by 26.9% from 0.0006098 to 0.0004458 while maintaining a C_T value of 0.00553 from a starting value of 0.00575. These optimization results are shown in Fig. 60. The optimized design features a blade tip with a sharper, downturned leading edge and a thinner, asymmetric section shape. The initial and final surface shapes with C_p contours are compared in Fig. 58. Note that the strong shock on the upper surface has been removed.

C. Adaptive Mesh Refinement

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

1. *Goal-oriented mesh adaptation*

To study the applicability of the goal-oriented mesh adaptation, a two-dimensional, Euler, supersonic inlet test case, in particular the Clemens experimental configuration,[94] was selected.

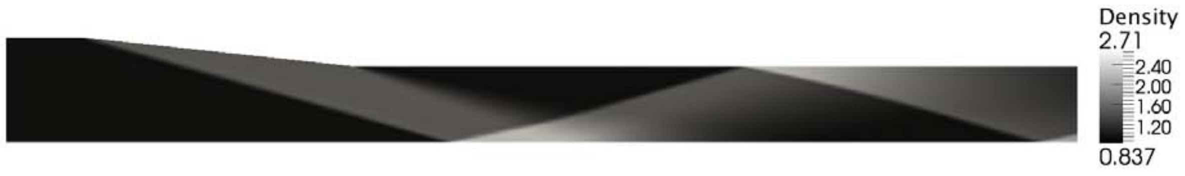


Figure 61. Density contours for nominal conditions (inflow Mach number 5.0 and ramp angle 6.0°).

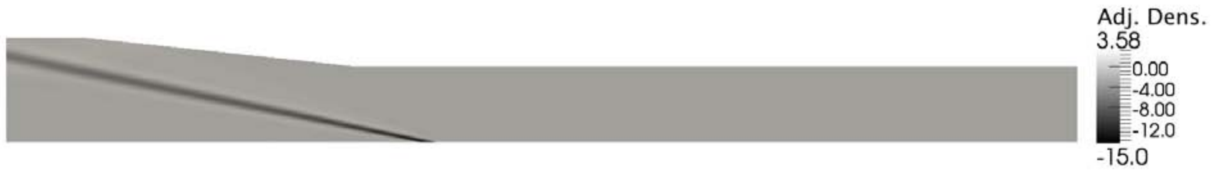


Figure 62. Density adjoint variable contours for nominal conditions and pressure sensor located on the lower wall.

The nominal conditions are an inlet Mach number of 5.0 and a ramp angle 6.0° (see Fig. 61). The computation (see Fig. 62) was performed using a second-order continuous adjoint formulation. In this case, we were interested in predicting accurately the pressure signature in an area of the lower wall that is located slightly downstream of the end of the inlet ramp.

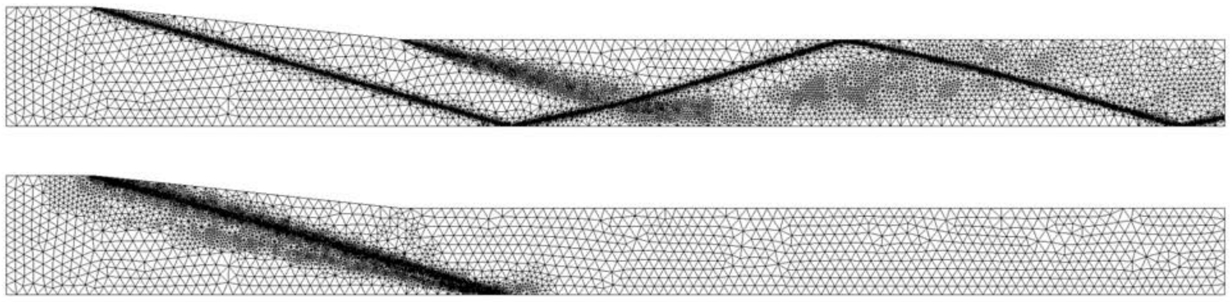


Figure 63. Grid adaptation using a gradient-based method (upper), adjoint-based computable error (lower).

In Fig. 63, the goal-oriented adaptation is compared with the gradient-based technique. Due to the supersonic flow, the goal-oriented technique only adapts the numerical grid upstream of the sensor location. On the other hand, the gradient-based methodology adapts the numerical grid along the entire shock train.

2. Engine propulsion effect adaptation

The gradient-based adaptation methodology was also tested with highly complex geometries. For example, the adaptation techniques were used to study the effects of propulsion integration for the Lockheed N+2 aircraft geometry ($M_\infty = 1.7$, angle of attack $2.1°$). Here the final target was to evaluate the pressure signature in the near-field, and two levels of solution adaptation were used to capture minor effects due to the propulsion.

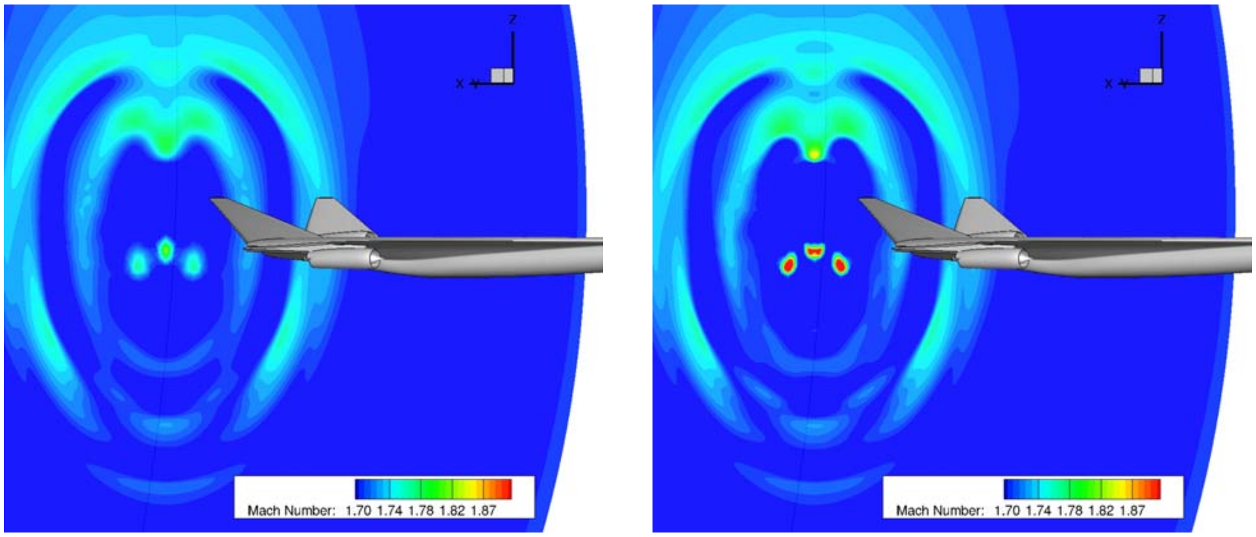

Z
X
Mach Number: 1.70 1.74 1.78 1.82 1.87
Z
X
Mach Number: 1.70 1.74 1.78 1.82 1.87

Figure 64. Gradient based mesh refinement for the Lockheed N+2 aircraft geometry ($M_\infty = 1.7$, AoA $2.1°$, with engine propulsion effects), showing the change in the Mach number contours.

In Fig. 64 and Fig. 65 the baseline and two levels of adaptation are shown. The baseline mesh has 1.3 million nodes, while the adapted grids have 1.5 and 1.8 million nodes, respectively.

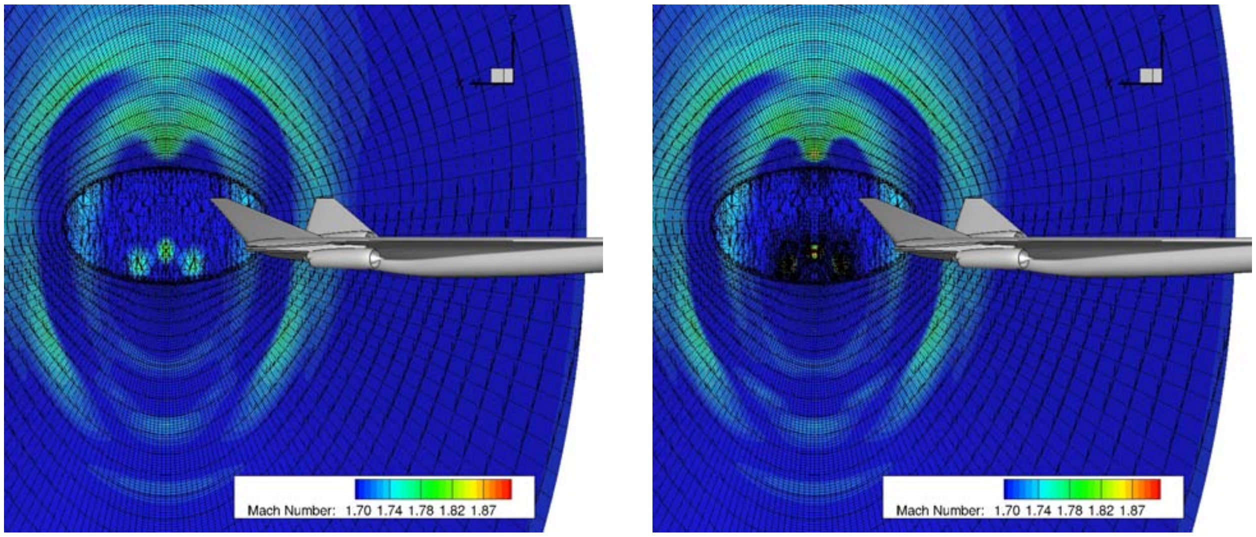

Z
Mach Number: 1.70 1.74 1.78 1.82 1.87
Z
Mach Number: 1.70 1.74 1.78 1.82 1.87

Figure 65. Gradient based mesh refinement for the Lockheed N+2 aircraft geometry ($M_\infty = 1.7$, AoA $2.1°$, with engine propulsion effects), showing the change in the grid.

The effect of the adaptation is also reflected in the near-field pressure signature, plotted at different azimuthal angles in Fig. 66, which shows minor differences.

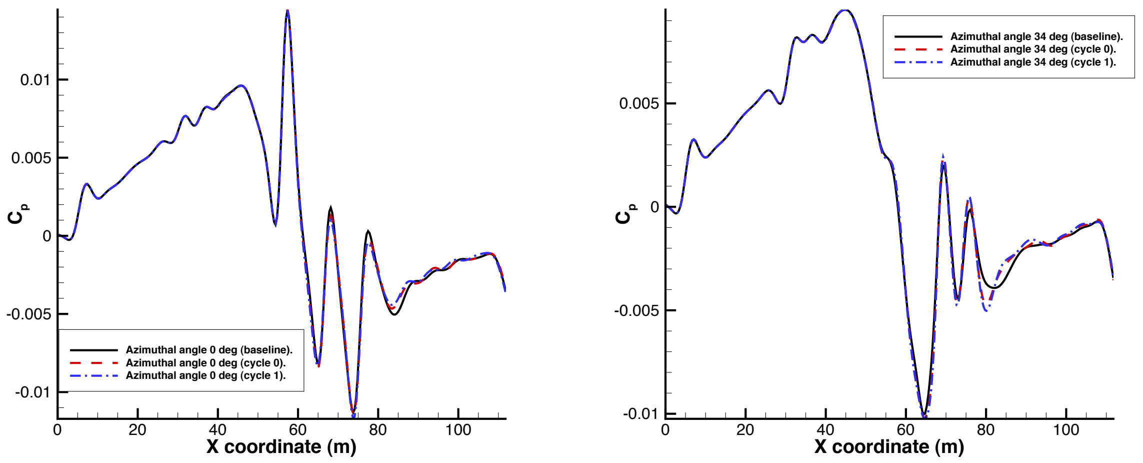
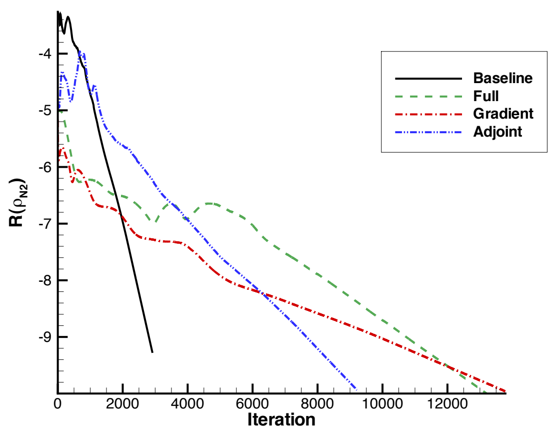


Figure 66. Adjoint-based mesh refinement for the RAM-C II hypersonic flight test experiment.

3. Plasma adaptation

The Mesh Adaptation Code within SU^2 is capable of handling large, multi-physics simulations due to the general formulation of the adaptation strategies. The results displayed here demonstrate that capability on a high Mach number flow in thermo-chemical non-equilibrium for a number of different strategies.



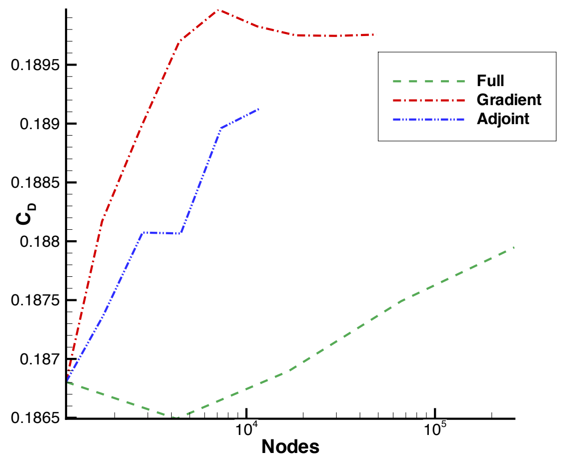


Figure 67. Log-reduction in density residual for the most highly adapted meshes.

Figure 68. C_D convergence for the adaptation schemes.

This particular application is suitable for mesh adaptation because accurately simulating high-speed gas flows requires grids of high quality. Post-shock conditions depend strongly on mesh resolution and must be adequately resolved for accurate force and energy predictions at domain boundaries. Furthermore, the inclusion of multiple chemical constituents and thermochemical non-equilibrium increases the size of the linear system and introduces stiff source terms, placing a premium on efficient solution strategies.

The problem considered here is based on the RAM-C II test article for the series of hypersonic flight tests[47] conducted at NASA to quantify electron number densities around entry vehicles. This is an axisymmetric body with a nose radius of $0.1524\,m$, a half-angle of $9°$, and a total length of $1.295\,\text{m}$. Freestream conditions

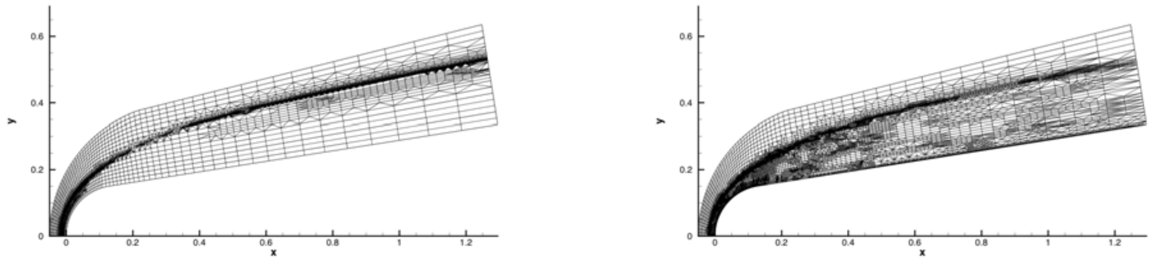


Figure 69. Adapted meshes: (left) gradient-based adapted, (right) adjoint-based adapted.

match those of Case 6 in the original test report and are summarized as follows: $M_\infty = 25.9$, $H = 71\,\mathrm{km}$, $T_\infty = 216\,\mathrm{K}$, and a Reynolds number of 6280.

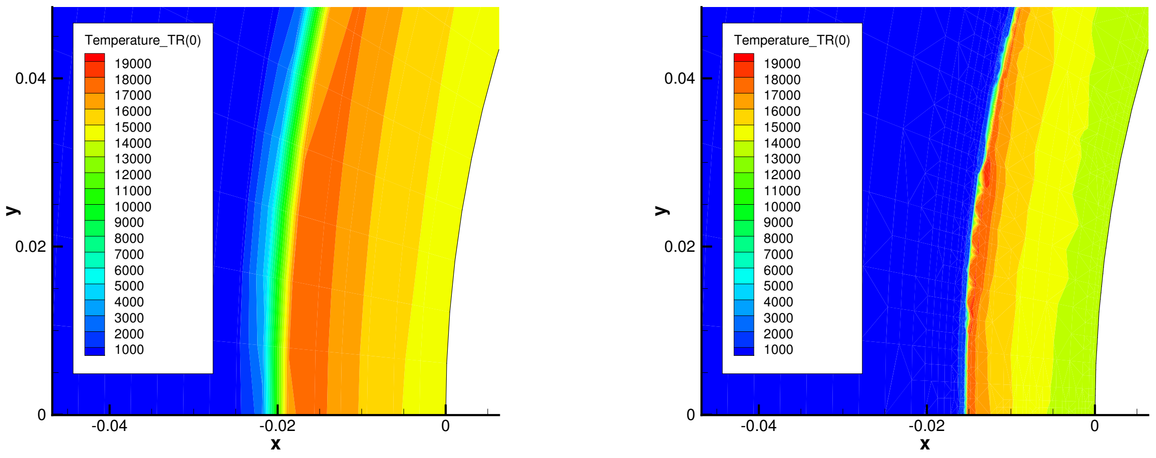


Figure 70. Effect on temperature distribution of adaptation behind the shock at the nose of the RAM-C II geometry: (left) baseline, (right) adjoint-based adapted.

The simulation uses an inviscid, two-species Nitrogen gas chemistry model at the specified freestream conditions, and the adjoint problem uses the drag coefficient as the objective function (first-order, Roe spatial discretization). The convergence history of the different grids used are shown in Fig. 67, and a comparison of the different strategies with the value of the drag coefficient is plotted against mesh size for each method in Fig. 68. Fig. 69 shows how the gradient- and adjoint-based approaches adapt the grid, and Fig. 70 shows the effect of adjoint-based adaptation on the temperature solution.

VI. Conclusions

This paper has presented a detailed overview of the objectives, implementation, and capabilities of the Stanford University Unstructured (SU^2) analysis and optimization suite. The suite can be used to analyze the behavior of problems governed by arbitrary PDEs that are discretized on arbitrarily-complex unstructured meshes. Moreover, SU^2 can be used to solve PDE-constrained optimization problems defined by the user. The suite takes advantage of modern programming techniques resulting in code that is portable, reusable, modular, and freely available through an open-source license.

In addition to describing many of the details of each component of the suite including their formulation and discretization, this paper presents up-to-date examples of the use of SU^2 for a variety of problems such as steady and unsteady Euler and RANS, multi-species and non-equilibrium flows, low-speed and supersonic simulations, or free surface formulations, to name a few. The solution of the corresponding adjoint systems for these problems provides sensitivity information that can be directly used for optimization, provided to uncertainty quantification techniques, or even used to construct surrogate models, and several examples were given in this article.

The result is an extensible framework that is available to the user community for further development. Two important considerations during the initial development of SU^2 have been to ensure (a) that the complete framework is available for research at locations and institutions that do not have the resources to create environments like this from scratch, and (b) that the resulting code base remains open source (so that its components can be peer-reviewed and improved upon) and available as long as it serves a useful purpose for the community.

Over the past year (since the suite was initially released), the software has been downloaded by more than $3,000$ institutions and individuals worldwide. The development team has engaged in many interactions with the user community that have led to improvements in the installation procedure, the documentation, and the validation of multiple test cases. Many suggestions for improvement have been made, and several groups around the world have expressed interest in developing new modules for the suite. We strongly encourage such efforts by members of our community and look forward to seeing many future developments incorporated into the code.

The development of SU^2 is supported by the Aerospace Design Laboratory at Stanford University, which will continue to host all major activities involving the software suite. For more information regarding the current status of SU^2, please visit our web page, su2.stanford.edu.

Acknowledgments

Francisco Palacios would like to acknowledge the support of the U.S. Department of Energy under the Predictive Science Academic Alliance Program (PSAAP), and also thank Prof. Jameson (Stanford) and Prof. Zuazua (BCAM) for helpful discussions and comments. Michael R. Colonno would like to acknowledge Aeroflightdynamics Directorate (AFDD) at NASA Ames Research Center. Aniket C. Aranake would like to acknowledge the support the U.S. Department of Defense, National Defense Science and Engineering Graduate (NDSEG) Fellowship. Sean R. Copeland would like to thank Mr. and Mrs. John Lillie for their generous contributions to the Stanford Graduate Fellowship program, through which this research was funded. Thomas D. Economon would like to acknowledge U.S. government support under and awarded by the U.S. Department of Defense, Air Force Office of Scientific Research, National Defense Science and Engineering Graduate (NDSEG) Fellowship, 32 CFR 168a. Amrita K. Lonkar would like to acknowledge the support from the Stanford Graduate Fellowship. Trent W. Lukaczyk would like to thank the support of the NASA Supersonics Project of the NASA Fundamental Aeronautics Program. Thomas W. R. Taylor would like to thank the support of the U.S. Department of Energy under the Predictive Science Academic Alliance Program (PSAAP). The SU^2 (v2.0) developers would also like to thank the following researchers: Karthik Duraisamy (Stanford), Jason Hicken (RPI), Carlos Castro (UPM), Alfonso Bueno (University of Oxford), Michael Buonanno (Lockheed-Martin Corporation), Rubén Pérez (Royal Military College of Canada), Gérald Carrier (ONERA), Kedar Naik (Stanford), Santiago Padrón (Stanford).

References

[1]Juan J. Alonso and Michael Colonno. Multidisciplinary optimization with applications to sonic-boom minimization. In *Annual Review of Fluid Mechanics*, volume 44, pages 505–526. 2012.

[2]W.K. Anderson and V. Venkatakrishnan. Aerodynamic design optimization on unstructured grids with a continuous adjoint formulation. *AIAA Paper*, 97-0643, 1997.

[3]J.D. Anderson Jr. *Hypersonic and High-Temperature Gas Dynamics.* AIAA Education Series, 2006.

[4]A. Baeza, C. Castro, F. Palacios, and E. Zuazua. 2-D Euler shape design on nonregular flows using adjoint Rankine-Hugoniot relations. *AIAA Journal*, 47(3), 2009.

[5]T.J. Barth. Aspect of unstructured grids and finite-volume solvers for the Euler and Navier-Stokes equations. In *Lecture Notes Presented at the VKI Lecture Series*, 1994 - 05, Rhode Saint Genese Begium, 2 1995. Von karman Institute for fluid dynamics.

[6]R. Biswas and R.C. Strawn. Tetrahedral and hexahedral mesh adaptation for cfd problems. *Applied Numerical Mathematics*, 26:135–151, 1998.

[7]A. Borzi. Introduction to multigrid methods. Technical report, Institut für Mathematik und Wissenschaftliches Rechnen (Karl-Franzens-Universität Graz), 2003.

[8]A. Bueno-Orovio, C. Castro, F. Palacios, and E. Zuazua. Continuous adjoint approach for the Spalart–Allmaras model in aerodynamic optimization. *AIAA Journal*, 50(3), 2012.

[9]J. Cahouet. Etude numerique et experimentale du probleme bidimensionnel de la resistance de vagues non-lineaire. Technical report, Technical report 185, Ecole Nationale Superieure de Techniques Avancees, 1984.

[10]G. V. Candler and R. W. MacCormack. Computation of weakly ionized hypersonic flows in thermochemical nonequilibrium. *Journal of Thermophysics and Heat Transfer*, 5(3):266–273, 1991.

[11]F. X. Caradonna and C. Tung. Experimental and analytical studies of a model helicopter rotor in hover. Technical Report NASA Technical Memorandum 81232 (NASA Ames Research Center, Moffett Field, CA), 1981.

[12]C. Castro, C. Lozano, F. Palacios, and E. Zuazua. A systematic continuous adjoint approach to viscous aerodynamic design on unstructured grids. *AIAA Journal*, 45(9):2125–2139, 2007.

[13]A. J. Chorin. A numerical method for solving incompressible viscous flow problems. *Journal of Computational Physics*, 137:118–125, 1997.

[14]M. R. Colonno, K. Naik, K. Duraisamy, and J. J. Alonso. An adjoint-based multidisciplinary optimization framework for rotorcraft systems. AIAA Paper 2012-5656, 2012.

[15]P. Cook, M. McDonald, and M. Firmin. Aerofoil RAE2822 pressure distributions, and boundary layer and wake measurements. Technical Report AGARD 138, 1979.

[16]S. R. Copeland, A. K. Lonkar, F. Palacios, and J. J. Alonso. Adjoint-based goal-oriented mesh adaptation for nonequilibrium hypersonic flows. *AIAA Paper, 51st Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition (Submitted for Publication), Grapevine, TX*, January 2013.

[17]G. F. Corliss, C. Faure, A. Griewank, and L. Hascoet. *Automatic Differentiation of Algorithms: From Simulation to Optimization.* Springer Verlag, New York, 2002.

[18]S. S. Davis. NACA 64A010 (NASA Ames model) oscillatory pitching, compendium of unsteady aerodynamic measurements. Technical Report AGARD, Rept. R-702, 1982.

[19]T. D. Economon, F. Palacios, and J. J. Alonso. A coupled-adjoint method for aerodynamic and aeroacoustic optimization. AIAA Paper 2012-5598, 2012.

[20]T. D. Economon, F. Palacios, and J. J. Alonso. Optimal shape design for open rotor blades. AIAA Paper 2012-3018, 2012.

[21]T. D. Economon, F. Palacios, and J. J. Alonso. Unsteady aerodynamic design on unstructured meshes with sliding interfaces. *AIAA Paper, 51st Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition (Submitted for Publication), Grapevine, TX*, January 2013.

[22]P. Eliasson. Edge, a Navier-Stokes solver for unstructured grids. Technical Report FOI-R-0298-SE, FOI Scientific Report, 2002.

[23]J. E. Ffowcs Williams and D. L. Hawkings. Sound generation by turbulence and surfaces in arbitrary motion. *Philosophical Transactions of the Royal Society of London*, A 342, pp.264-321, 1969.

[24]M. B. Giles and N. A. Pierce. Adjoint error correction for integral outputs. Error Estimation and Adaptive Discretization Methods in Computational Fluid Dynamics, 2002.

[25]A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation (2nd ed.).* Society for Industrial and Applied Mathematics (SIAM), 2008.

[26]L. Hascoët and V. Pascual. Tapenade 2.1 user's guide. Technical Report 0300, INRIA, http://www.inria.fr/rrrt/rt-0300.html, 2004.

[27]R.M. Hicks and P.A. Henne. Wing design by numerical optimization. *Journal of Aircraft*, 15:407–412, 1978.

[28]C. Hirsch. *Numerical Computation of Internal and External Flows.* Wiley, New York, 1984.

[29]D. G. Holmes and S. S. Tong. A three-dimensional Euler solver for turbomachinery blade rows. *Journal of Engineering for Gas Turbines and Power*, 107, 1985.

[30]Kitware Inc. *VTK User's Guide Version 11.* Kitware Inc., 2010.

[31]J. J. Donea, A. Huerta, J.-Ph. Ponthot, and A. Rodriguez-Ferran. *Arbitrary Lagrangian-Eulerian Methods in Encyclopedia of Computational Mechanics.* John Wiley and Sons, 2004.

[32]A. Jameson. Aerodynamic design via control theory. *Journal of Scientific Computing*, 3:233–260, 1988.

[33]A Jameson. Time dependent calculations using multigrid, with applications to unsteady flows past airfoils and wings. *AIAA Paper*, 91-1596, 1991.

[34]A. Jameson. 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*, 4:171–218, 1995.

[35]A. Jameson. 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*, 5:1–38, 1995.

[36]A. Jameson. A perspective on computational algorithms for aerodynamic analysis and design. *Progress in Aerospace Sciences*, 37:197–243, 2001.

[37]A. Jameson. Aerodynamic shape optimization using the adjoint method. In *Lecture Notes Presented at the VKI Lecture Series*, Rhode Saint Genese Begium, 2 2003. Von karman Institute for fluid dynamics.

[38]A. Jameson and S. Kim. Reduction of the adjoint gradient formula for aerodynamic shape optimization problems. *AIAA Journal*, 41(11):2114–2129, 2003.

[39]A. Jameson, L. Martinelli, and F. Grasso. A multigrid method for the Navier-Stokes equations. *AIAA Paper*, 86-0208, 1986.

[40]A. Jameson and S. Schenectady. An assessment of dual-time stepping, time spectral and artificial compressibility based numerical algorithms for unsteady flow with applications to flapping wings. *AIAA Paper*, 20094273, 2009.

[41]A. Jameson, W. Schmidt, and E. Turkel. Numerical solution of the euler equations by finite volume methods using runge-kutta time stepping schemes. *AIAA Paper*, 81-1259, 1981.

[42]A. Jameson and Y. Seokkwan. Lower-Upper implicit schemes with multiple grids for the euler equations. *AIAA Journal*, 25(7):929–935, 1987.

[43]A. Jameson, S. Sriram, and L. Martinelli. A continuous adjoint method for unstructured grids. *AIAA Paper*, 2003-3955, 2003.

[44]A. Jameson, S. Sriram, L. Martinelli, and B. Haimes. Aerodynamic shape optimization of complete aircraft configurations using unstructured grids. *AIAA Paper*, 2004-533, 2004.

[45]A. Jameson and E. Turkel. Implicit schemes and LU-decompositions. *Mathematics of Computation*, 37:385–397, 1981.

[46]E. Jones, T. Oliphant, P. Peterson, et al. SciPy: Open source scientific tools for Python, 2001–.

[47]W. L. Jones and A. E. Cross. Electrostatic probe measurements of plasma parameters for two reentry flight experiments at 25,000 feet per second. Nasa tn d-6617, Washington, DC, February 1972.

[48]G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/~metis, 2009.

[49]S. Kim, J.J. Alonso, and A. Jameson. Design optimization of high-lift configurations using a viscous continuous adjoint method. *AIAA Paper*, 2002-0844, 2002.

[50]L.D. Landau and E.M. Lifshitz. *Fluid Mechanics (2nd Edition)*. Pergamon Press, 1993.

[51]R. B. Langtry and F. R. Menter. Correlation-based transition modeling for unstructured parallelized computational fluid dynamics codes. *AIAA Journal*, 47(12), 2009.

[52]J. H. Lee. Basic governing equations for the flight regimes of aeroassisted orbital transfer vehicles. In H. F. Nelson, editor, *Thermal Design of Aeroassisted Orbital Transfer Vehicles*, volume 96, pages 3–53. AIAA, New York, 1985.

[53]R. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Univesity Press, 2002.

[54]A. K. Lonkar, F. Palacios, R. W. MacCormack, and J. J. Alonso. Multidimensional simulation of plasma in argon through a shock in hypersonic flow. *43rd AIAA Thermodynamics Conference 25-28 June 2012, New Orleans, Louisiana*, (3105), 2012.

[55]T. Lukaczyk, F. Palacios, and J. J. Alonso. Response surface methodologies for low-boom supersonic aircraft design using equivalent area distributions. AIAA Paper 2012-5705, 2012.

[56]T. Lukaczyk, F. Palacios, and J. J. Alonso. Managing gradient inaccuracies while enhancing optimal shape design methods. *AIAA Paper, 51st Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition (Submitted for Publication), Grapevine, TX*, January 2013.

[57]R. W. MacCormack, D. D'Ambrosio, D. Giordano, J. K. Lee, and T. Kim. Plasmadynamic simulations with strong shock waves. *AIAA Paper, 42nd Plasmadynamics and Lasers Conference, Honolulu, HI,*, 2011-3921, June 2011.

[58]D. J. Mavriplis. Multigrid techniques for unstructured meshes. Technical report, Institute for computer applications on science and engineering (ICASE), 1995.

[59]D. J. Mavriplis. On convergence acceleration techniques for unstructured meshes. Technical report, Institute for computer applications on science and engineering (ICASE), 1998.

[60]D.J. Mavriplis. Discrete adjoint-based approach for optimization problems on three-dimensional unstructured meshes. *AIAA Journal*, 45(4):740–750, 2007.

[61]S. Medida and J. Baeder. Numerical prediction of static and dynamic stall phenomena using the $\gamma - \overline{Re_{\theta t}}$ transition model. In *American Helicopter Society* 67th *Annual Forum*, Virginia Beach, VA, May 2011.

[62]F.R. Menter. Zonal two equation $k - \omega$, turbulence models for aerodynamic flows. *AIAA Paper*, 93-2906, 1993.

[63]B. Mohammadi and O. Pironneau. Shape optimization in fluid mechanics. *Annual Rev. Fluids Mechanics*, 36:255–279, 2004.

[64]S.K. Nadarajah and A. Jameson. A comparison of the continuous and discrete adjoint approach to automatic aerodynamic optimization. *AIAA Paper*, 2000-0667, 2000.

[65]R. J. Nowak and M. C. Yuen. Heat transfer to a hemispherical body in a supersonic argon plasma. *AIAA Journal*, 2(2):1463–1464, 1973.

[66]F. Palacios and J. J. Alonso. New convergence acceleration techniques in the Joe code. Technical report, Center for Turbulence Research, Annual Research Brief 2011, 2011.

[67]F. Palacios, J. J. Alonso, M. Colonno, J. Hicken, and T. Lukaczyk. Adjoint-based method for supersonic aircraft design using equivalent area distributions. AIAA Paper 2012-0269, 2012.

[68]F. Palacios, J. J. Alonso, and A. Jameson. Shape sensitivity of free-surface interfaces using a level set methodology. AIAA Paper 2012-3341, 2012.

[69]F. Palacios, K. Duraisamy, J. J. Alonso, and E. Zuazua. Robust grid adaptation for efficient uncertainty quantification. *AIAA Journal*, 50(7):1538–1546, 2012.

[70]C. Park. *Nonequilibrium Hypersonic Aerothermodynamics*. Wiley, New York, NY, 1990.

[71]N.A. Pierce and M.B. Giles. Preconditioned multigrid methods for compressible flow calculations on stretched meshes. *J. Comput. Phys.*, 136:425–445, 1997.

[72]O. Pironneau. On optimum design in fluid mechanics. *J. Fluid Mech.*, 64:97–110, 1974.

[73]A. Quarteroni and A. Valli. *Numerical aproximation of partial differential equations*, volume 23 of *Springer series in computational mathematics*. Springer-Verlag Berlin Heidelberg New York, 1997.

[74]C.E. Rasmussen and C.K.I. Williams. *Gaussian Processes for Machine Learning*, pages 13–30. MIT Press, Cambridge, MA, 2006.

[75]P.L. Roe. Approximate riemann solvers, parameter vectors, and difference schemes. *Journal of Computational Physics*, 43:357–372, 1981.

[76]C. Rumsey and et al. The menter shear stress turbulence model. World Wide Web electronic publication, 2012.

[77]Y. Saad and M.H. Schultz. GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM J. Sci. Stat. Comput.*, 7:856–869, 1986.

[78]J. A. Samareh. Aerodynamic shape optimization based on free-form deformation. *AIAA Paper*, 2004-4630, 2004.

[79]V. Schmitt and F. Charpin. Pressure distributions on the onera-m6-wing at transonic mach numbers. Technical Report AGARD, Report AR-138, 1979.

[80]T. W. Sederberg and S. R. Parry. Free-form deformation of solid geometric models. *Proceedings of SIGGRAPH 89 (Computer Graphics)*, 20(4):151–159, 1986.

[81]J.A. Sethian. *Level set method and fast marching methods*. Cambridge Monographs in Applied and Computational Mathematics. Cambridge University Press, 2002.

[82]J.-P. Sokolowski, J. Zolesio. *Introduction to shape optimization*. Springer Verlag, New York, 1991.

[83]O. Soto, R. Lohner, and F. Camelli. A linelet preconditioner for incompressible flow solvers. *Int. J. Numer. Meth. Heat Fluid Flow*, 13(1):133–147, 2003.

[84]P Spalart and S. Allmaras. A one-equation turbulence model for aerodynamic flows. *AIAA Paper*, 92-0439, 1992.

[85]M. Sussman, P. Smereka, and . Osher. A level set approach for computing solutions to incompressible two-phase flow. *Journal of Computational Physics*, 114(1):146–159, 1997.

[86]T. W. R. Taylor, F. Palacios, K. Duraisamy, and J. J. Alonso. Towards a hybrid adjoint approach for arbitrarily complex partial differential equations. AIAA Paper 2012-3342, 2012.

[87]E. F. Toro. *Riemann Solvers and Numerical Methods for Fluid Dynamics: a Practical Introduction*. Springer-Verlag, 1999.

[88]E. Turkel. Preconditioning techniques in fluid dynamics. Technical report, School of Mathematical Sciences, Tel-Aviv University, 1999.

[89]E. Turkel, V. N. Vatsa, and R. Radespiel. Preconditioning methods for low-speed flows. *AIAA*, 1996.

[90]H. A. Van der Vorst. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM J. Sci. and Stat. Comput.*, 13(2), 1992.

[91]B. van Leer. Towards the ultimate conservative difference scheme v. a second order sequel to godunov's method. *J. Com. Phys.*, 32, 1979.

[92]W. Vincenti and C. H. Kruger Jr. *Introduction to Physical Gas Dynamics*. Krieger, 1965.

[93]C. Viozat. Implicit upwind schemes for low mach number compressible flows. Technical report, Institut National De Recherche En Informatique En Automatique, 1997.

[94]J. L. Wagner, A. Valdivia, K. B. Yuceil, N. T. Clemens, and D. S. Dolling. An experimental investigation of supersonic inlet unstart. AIAA Paper 2007-4352, 2007.

[95]J.M. Weiss, J.P. Maruszewski, and A.S. Wayne. Implicit solution of the Navier-Stokes equation on unstructured meshes. *AIAA Journal*, 97-2103, 1997.

[96]P. Wesseling. *Principles of computational fluid dynamics*, volume 29 of *Springer series in computational mathematics*. Springer-Verlag Berlin Heidelberg New York, 2000.

[97]F.M. White. *Viscous Fluid Flow*. McGraw Hill Inc., 1974.

[98]D.C. Wilcox. *Turbulence Modeling for CFD*. 2nd Ed., DCW Industries, Inc., 1998.

[99]S. Yoon and A. Jameson. Lower-Upper Symmetric-Gauss-Seidel method for the Euler and Navier-Stokes equations. *AIAA Journal*, 26(9), 1988.

[100]O.C. Zienkiewicz and R.L. Taylor. *Finite Element Method, 6th ed., Vols. 1, 2 and 3*. Elsevier, Oxford, UK, 2005.