Supercomputing Frontiers and Innovations

2022, Vol. 9, No. 4

Scope

- Future generation supercomputer architectures
- Exascale computing
- Parallel programming models, interfaces, languages, libraries, and tools
- Supercomputer applications and algorithms
- Novel approaches to computing targeted to solve intractable problems
- Convergence of high performance computing, machine learning and big data technologies
- Distributed operating systems and virtualization for highly scalable computing
- Management, administration, and monitoring of supercomputer systems
- Mass storage systems, protocols, and allocation
- Power consumption minimization for supercomputing systems
- Resilience, reliability, and fault tolerance for future generation highly parallel computing systems
- Scientific visualization in supercomputing environments
- Education in high performance computing and computational science

Contents

DOI: 10.14529/jsfi220401

A Comparison of Different Approaches for Predicting Transonic Buffet Onset on Infinite Swept Wings

Kirill V. Belyaev[1], ***Andrey V. Garbaruk***[1], ***Sergey V. Kravchenko***[2], ***Michael K. Strelets***[1]

A comparative study is performed on three different approaches for prediction of transonic buffet onset on infinite swept wings. All three approaches are based on the unsteady Reynolds-averaged Navier–Stokes (URANS) equations, and include: quasi-3D and fully-3D global stability analysis of the corresponding steady 2.5D RANS solutions and direct numerical solution of the 3D URANS equations. The results are presented for an infinite swept wing based on the ONERA OAT15A airfoil section. The quasi-3D stability analysis is shown to be accurate and most efficient and, thus, is best suited for this spanwise-uniform flow. The fully-3D stability analysis ensures the same accuracy, provided that the grid-step in the spanwise direction is sufficiently small. It is much more demanding in terms of computer memory but can be extended to more-general wing configurations. Good agreement is observed between the three approaches in terms of critical conditions for buffet onset and the instability growth characteristics, providing a cross-validation of the methods and an assessment of their computational demands.

Keywords: transonic buffet onset, infinite swept wing, global stability analysis, quasi-3D and fully-3D approaches, direct URANS solution.

Introduction

Transonic buffet phenomenon is characterized by self-sustained oscillations of the shockwave that forms on the upper surface of airplane wings. At Mach numbers typical of cruise flight for commercial airliners ($M = 0.75 - 0.9$), these shock oscillations turn out to be rather intensive over a wide range of angles of attack and lead to large airplane-level normal accelerations. This can limit the maximum allowable angle of attack, and, therefore, the maximum lift and also the flight altitude. Hence, a determination of the conditions for the buffet onset and its characteristics is essential for airplane design, and it is not surprising that different aspects of this phenomenon (its mechanism, critical values of the onset Mach number and angle of attack, and post-onset unsteady characteristics) have been investigated in a very large number of experimental and numerical studies (see the review of Giannelis et al. [6], and more recent papers [2, 3, 8, 12–17, 21, 23]).

According to the experimental studies, the shock-wave oscillations observed in transonic-wing flow after buffet onset are characterized by frequencies that are much smaller than the frequencies associated with turbulent fluctuations typical of high-Reynolds-number aerodynamic flows. This justifies modeling of the buffeting flows and predicting the buffet onset based on the direct time-accurate numerical integration of the unsteady Reynolds-averaged Navier–Stokes equations (URANS). Along with this, it is now generally accepted that the buffet phenomenon is explained by the onset of a global instability in transonic-wing flows at supercritical values of the Mach number and angle of attack [6]. This opens the possibility of using an alternative approach to predicting the buffet onset, namely, applying global stability analysis to steady RANS solutions. The applicability of this approach was first demonstrated for two-dimensional

[1]Peter the Great Saint-Petersburg Polytechnic University, Saint-Petersburg, Russian Federation
[2]The Boeing Company, Chicago, USA

(2D) airfoil buffet [1, 4], and the approach is now widely used in the investigation of buffeting flows (see, e.g. [2, 3, 14, 15, 23]).

Extensions of the approach to quasi-three-dimensional (q-3D) flows for application to infinite swept wings have been presented in the works [2, 14, 15]. Meanwhile, fully-three-dimensional (3D) stability analysis has been used to analyze a swept tapered wing [23]. More recently, He and Timme [8] used the fully-3D approach to analyze an infinite swept wing. This shows the linkage between the discrete modes of the fully-3D analysis and the continuous band of modes predicted by the q-3D analysis. The work of He and Timme [8] also considered the effects of a stationary mode as part of the baseflow for the onset of oscillations on an unswept wing. This is possible with the fully-3D analysis, but not the q-3D approach.

Here, we provide a quantitative comparison between the q-3D and fully-3D stability approaches for the infinite swept wing, along with comparisons to the unsteady simulations. These three approaches involve independent formulations, but all flow variables and turbulence-model parameters are matched for the comparisons, allowing an assessment of both the accuracy and the efficiency for varying spanwise representations.

In this article, the comparisons are carried out for an infinite swept wing based on the ONERA OAT15A airfoil section [10]. First, in Section 1, we briefly present an overview of the three approaches. Then, in Section 2, we discuss the basic flow conditions and some of the numerical aspects of the performed computations. In Section 3, results are presented and discussed. Finally, in Conclusion, we summarize the major findings of the study.

1. Overview of the Considered Approaches

Steady flows past an infinite swept wing are characterized by uniformity in the spanwise directions z, with flow variation dependent only on x and y (see Fig. 1). Hence, such flows may be computed by numerical integration of the RANS equations in which the spanwise derivatives are set zero, that is, within an effectively 2D problem statement. In order to distinguish this type of flow from the purely-2D and fully-3D flows it is often referred to as 2.5D flow (with flow in the z direction, but $\partial/\partial z = 0$).

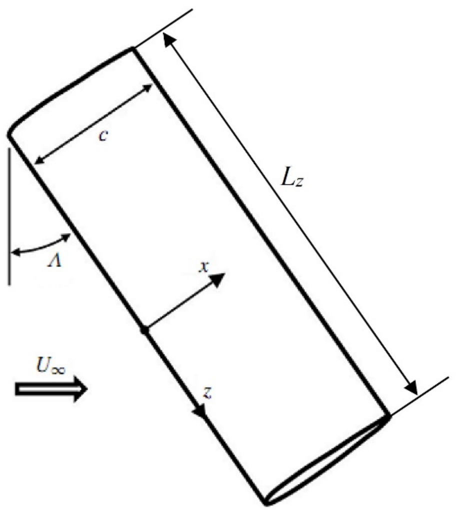


Figure 1. Schematic of section of infinite swept wing with the length L_z

The critical values of the angle of attack and Mach number corresponding to transonic buffet onset in the 2.5D flows may be determined using any of the three approaches based on the RANS equations. The first approach involves the direct numerical integration of the full 3D URANS equations for a wing section with a span length L_z (a free parameter of the problem) using periodic boundary conditions in the z-direction and a steady solution of the 2.5D RANS equations as initial conditions. For subcritical values of the angle of attack, $\alpha < \alpha_{crit}$, corresponding to a given Mach number, the URANS solution does not change in time, whereas for supercritical values $\alpha > \alpha_{crit}$, the solution is characterized by exponential growth of flow disturbances (deviations from the initial conditions). Thus, a series of 3D URANS computations at different values of α and L_z provides the critical buffet-onset conditions and the corresponding post-critical growth rate and frequency of the disturbances, based on post-processing of the simulation time-histories in the linear stage of the solution. The spanwise domain length L_z selects the admissible spanwise wavelengths for the disturbance, so multiple values should be considered. Along with this, based on the developed non-linear stage of the solution, one can compute the unsteady and mean local and integral aerodynamic characteristics of the buffeting flow, e.g. the lift and drag of the wing. This approach is the most general, but it demands large computational resources needed for performing multiple 3D unsteady simulations. Additionally, when close to critical buffet-onset conditions where disturbances grow or decay very slowly with time, the simulations require extremely long time samples and results can be highly sensitive to numerics.

Two alternative approaches for investigating buffet-onset conditions are based on global stability analysis of the 2.5D steady RANS solutions at different angles of attack. These approaches employ linear stability equations for describing the disturbances, which are derived by decomposing the total flow field into a steady baseflow and a small superimposed perturbation. Note that in the presence of shock waves, this is, strictly speaking, impossible, but still seems to be justified for the numerical solutions, if the perturbations are sufficiently small not to change the set of the grid cells "occupied" by the shock (i.e., if the shock remains effectively still). After introducing the decomposition into the 3D URANS equations with associated boundary conditions, and canceling terms which describe the steady RANS solution, the equations are linearized to provide a system of equations for the small 3D disturbances. Nontrivial solutions to this system are found by solving an eigenvalue problem for the corresponding linear differential operator [4]. The real and imaginary parts of the complex eigenvalues represent the frequency and the growth/decay rate (depending of the sign), respectively. The associated eigenfunction provides the spatial distribution for the disturbance. Hence, in order to answer the question on whether the considered steady solution is stable or unstable in the framework of these approaches, it is sufficient to find the eigenvalue with the maximum growth rate. If this value turns out to be negative, the steady solution is stable and otherwise it is unstable.

The first of the stability approaches, the q-3D approach, uses the spanwise invariance of the steady baseflow, and decomposes the disturbance into Fourier modes in z. Each mode is characterized by a spanwise wavelength λ_z, defined by the wave number β ($\lambda_z/c = 2\pi/|\beta|$), which is a free parameter of the q-3D stability problem. As a result of the Fourier decomposition in z, the q-3D eigenmodes are two-dimensional and sometimes referred to as "biglobal" [22]. The computational cost for this approach (both in CPU time and, especially the computer memory) is much less than for the URANS simulations or the fully-3D stability analysis described below. Not surprisingly, this method is widely used in the literature (see, e.g. [2, 3, 14]). However, for problems in which the steady baseflow is not strictly uniform in the spanwise direction, the q-3D formulation is only an approximation.

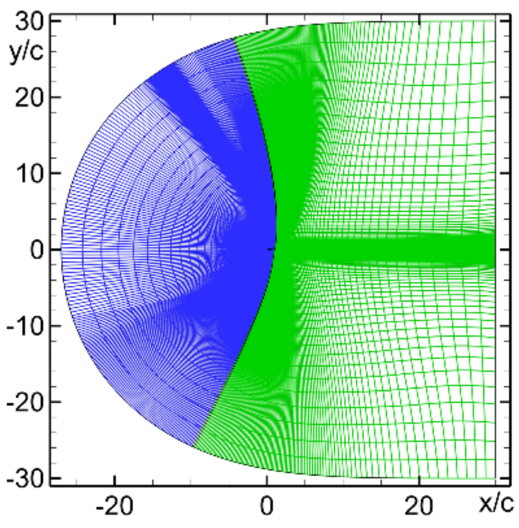
Thus, a more-general form of the stability problem is given by the fully-3D approach. Rather than introducing a Fourier decomposition in z, this approach considers a 3D eigenmode extending over the spanwise domain L_z. This is sometimes referred to as "triglobal" stability analysis [22]. In this formulation, the spanwise wavelength is set by the domain size L_z, just as it is in the URANS simulations. Note that in principle (neglecting any difference in the discretization errors of the URANS equations and the equations for the disturbances), predictions of the critical values for buffet onset based on the fully-3D stability analysis should coincide with those predicted based on the 3D URANS simulations, provided that the latter are started from the baseflow with superimposed disturbances of a very small amplitude (see below). However, computational resources needed for implementation of these two approaches are quite different: the 3D stability analysis is much less expensive than 3D URANS in terms of the CPU time, but is much more demanding in terms of computer memory, which strongly limits the size of the computational grids. For the current analysis of a 2.5D baseflow, this fully-3D analysis can be directly compared to the q-3D analysis, allowing an independent validation of each approach.

2. Flow Regimes and Numerical Aspects of Computations

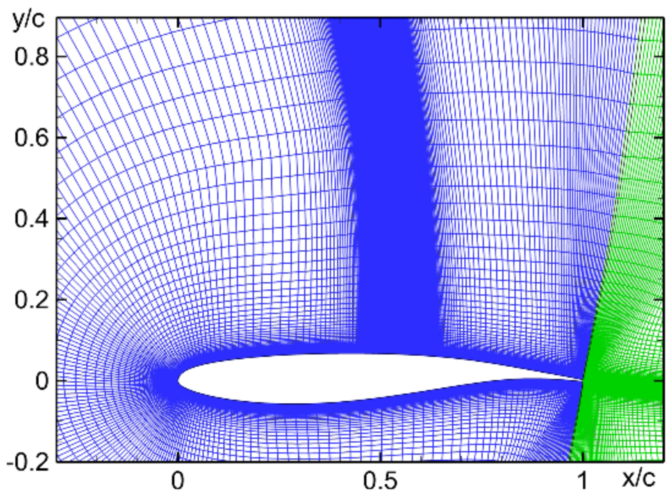
Specific computations performed in the present work are carried out for an infinite-span wing based on the supercritical airfoil OAT15A at different angles of attack α varying in the range from $2.75°$ up to $3°$ and fixed values of the wing sweep angle $\Lambda = 30°$, Reynolds number $Re_n = U_n c/\nu = 3 \cdot 10^6$ and Mach number $M_n = U_n/a_\infty = 0.73$. Nondimensionalization is based on the normal to the wing leading edge component of the free-stream velocity $U_n = U_\infty cos\Lambda$. The boundary-layer flow is assumed to be fully turbulent and is modeled using the Spalart–Allmaras turbulence model [19] with the Compressibility Correction [20] (SA CC model), which has been successfully used in many transonic buffeting flows (see, e.g. [1–4, 14]).

Computations are carried out with the in-house computer codes developed by the authors. In particular, the compressible steady and unsteady RANS equations are solved using the code of Numerical Turbulence Simulation (NTS) [18], and the algorithms implemented for solutions of the stability problems are presented in detail in [1, 2, 4]. The NTS code accepts structured multi-block overset grids. Computational domain and grid are divided into a set of grid blocks, taking into consideration both the flow geometry and effective usage of computers with large amount of nodes/cores. For massively parallel computations, the code employs a so-called hybrid Message Passing Interface (MPI)/Open Multi Processing (Open MP) parallelization scheme, which ensures a high efficiency of simulations on very large computational grids required for 3D URANS and scale-resolving simulations (see, e/g. [7]). Within this approach, both MPI libraries (with distributed memory technologies) and Open MP libraries (with shared memory technologies) are used for parallelization of computations in different grid blocks (or set of grid blocks). This strategy is very flexible and is easy to adapt to computers with different architecture by manually varying a set of managing parameters. The code has been thoroughly validated by comparisons with the solutions of a wide range of aerodynamic and stability problems and experimental data available in the literature.

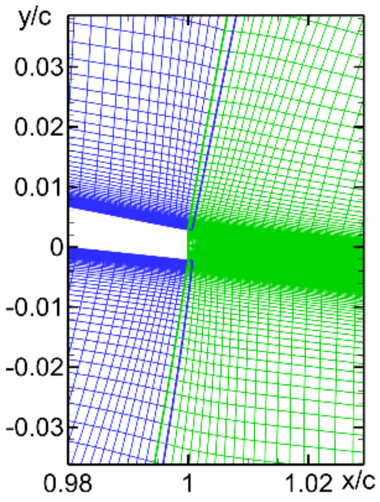
The computational domain and the grid in the XY-plane used for the computations are shown in Fig. 2. The domain has a radius of about $30c$, which, along with the use of the characteristic non-reflecting boundary conditions with the Riemann invariants computed by the free-stream flow parameters, ensures an adequate representation of the transonic flow past the wing. The structured computational grid has two overlapping blocks, shown in Fig. 2 by blue



(a) Entire domain



(b) Zoomed in view in the vicinity of the wing



(c) Zoomed in view in the vicinity of the blunt trailing edge

Figure 2. Computational domain and two-block grid in XY plane

and green grid lines. The main (blue) block is a C-grid with 343 $\times$ 140 cells, and the second (green) block is an H-grid with 97 $\times$ 374 cells. As a result, the total cell count of the grid in the XY-plane is about 85,000. In the streamwise direction the grid-step was reduced by a factor of ten over a wide area encompassing the shock location (in this area $\Delta x/c \approx 0.002$). The step in the wall-normal direction was decreased toward the wing surface so that the size of the first near-wall cell in the coordinates of the law of the wall $\Delta^+_{y,1} = \Delta_{y,1} u_\tau/\nu$ ($u_\tau = \sqrt{\tau_w/\rho}$ is the friction velocity) is less than 1.0.

The span length of the computational domain L_z used in the 3D URANS (this size defines the maximum wavelength of the solution) was set equal to $2c$, and the grid spacing in the z-direction Δz is uniform and equal to $0.005c$, which ensures sufficient resolution (not less than 20 cells per wavelength) of disturbances with length $\lambda_z > 0.1c$ (as will be shown shortly, for all the considered flows the most unstable modes have wavelengths within this range). As a result, the number of spanwise grid planes is $N_z = 400$, and the total cell count for the 3D URANS-simulation grid is around 35 million.

Time-integration of the URANS equations in the NTS code is performed with the use of the implicit three-layer backward scheme of second-order accuracy with the time-step $\Delta t = 10^{-3} \cdot c/U_n$, which ensures the Courant-Friedrichs-Lewy criterion $CFL < 1.0$ everywhere except for the close vicinity of the wing surface with extremely small wall-normal steps Δ_y.

The flow-fields used to initialize the unsteady simulations are chosen based on the angle of attack. Particularly, for $\alpha \geq 2.9^\circ$ (i.e., well above the critical angle of attack for buffet onset, see Fig. 10 below), the simulations are initiated with steady RANS solutions. At the lower α values, the initial fields are obtained by superposing the corresponding steady solutions with eigenvectors from the stability analysis with small (10^{-6}) amplitude. For the simulation at $\alpha = 2.8^\circ$ (stable according to URANS), this is the only way to determine the negative growth-rate (decay rate) in the URANS framework. At $\alpha = 2.85^\circ$, close to the URANS critical value, this approach offers a huge savings in CPU time; using only the steady RANS initial field, the time sample $T = c/U_n$ needed for development of the instability and extraction of the growth-rate would be an order of magnitude larger.

In principle, the fully-3D stability analysis provides simultaneous results for disturbances to the steady RANS solutions with wavelengths $\lambda_z = L_z/m$, where m is an arbitrary natural number. However, as mentioned above, this approach demands very large computer memory and, hence, imposes severe restrictions on the size of the computational grid. Particularly, with the memory available for the computations performed in the present study, the maximum affordable grid size for the fully-3D stability analysis is about 2 million cells. With the XY grid of 85,000 cells, this allows a maximum number of cells in the spanwise direction of $N_z = 22$. Thus, for a single solution only the disturbances at the maximum wavelength $\lambda_z = L_z$ may be analyzed reliably (i.e., with greater than 20 cells per wavelength). In order to consider different wavelengths, instead of using one computation in the domain with $L_z/c = 2$, as in the 3D URANS simulations, a series of 3D stability analyses are carried out with a fixed number of $N_z = 22$ cells in the spanwise direction, but with L_z/c varying over the range 0.5 up to 25. This ensured that the stability analysis only considered well resolved disturbances, i.e., those having the maximum wavelength $\lambda_z = L_z$ or the wave number $\beta = 2\pi/L_z$.

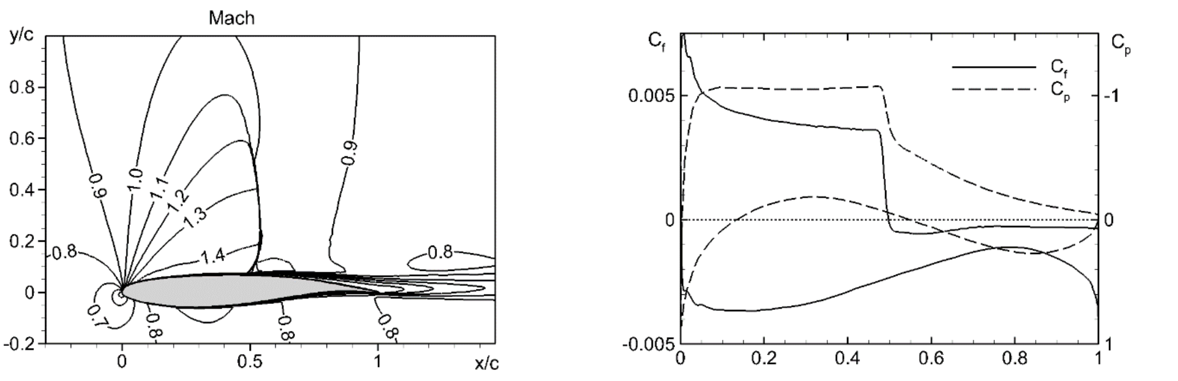
Finally, the q-3D stability analysis is performed using the same XY grid as the URANS simulations and the fully-3D stability analysis. Calculations are conducted for different values of β (a free parameter within this approach), varying in the range from $2\pi/25$ up to $2\pi/0.5$, i.e., in the range corresponding to the wavelengths addressed in the fully-3D analysis.

3. Results and Discussion

In order to show the basic character of the instability for transonic flows past an infinite swept wing, we first present results from the q-3D stability analysis for the considered flow at $\alpha = 2.95^\circ$. An extensive discussion of this problem is available in earlier papers [2, 14]. Building on this example, the primary results are then presented for the direct comparison of predictions using the three different approaches: q-3D stability analysis, fully-3D stability analysis, and 3D URANS simulations.

3.1. Baseflow and Q-3D Stability Results

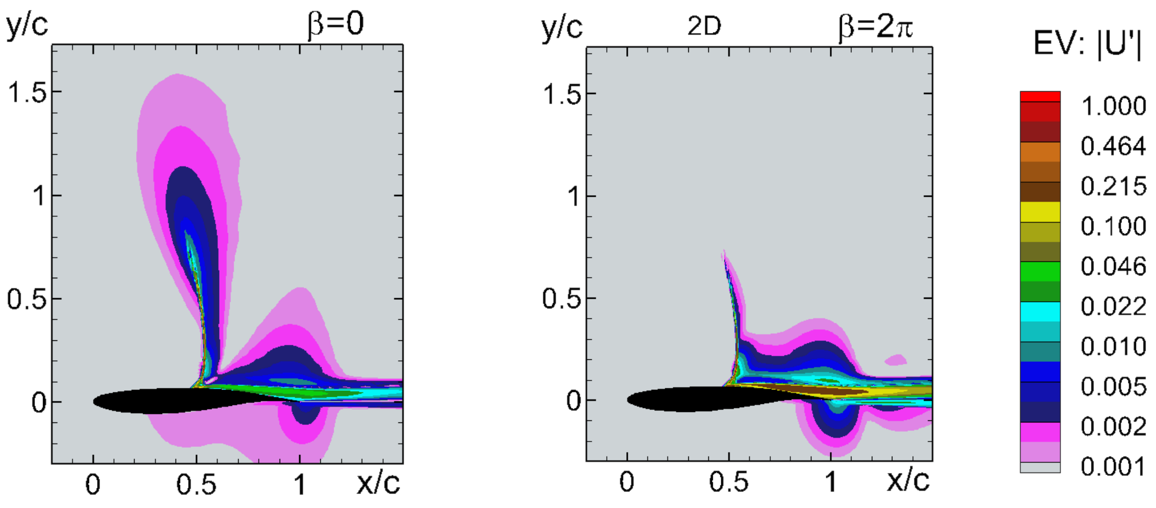
Figure 3 illustrates major features of the steady 2.5D RANS solutions used for the stability analysis. It shows contours of the Mach number in XY-plane and streamwise distributions of the skin-friction and pressure coefficients C_f and C_P over the wing surface at $\alpha = 2.95°$, as an example. One can see that the flow is characterized by the formation of a shock closing the supersonic region above the suction side of the wing. This shock induces separation of the boundary layer, which extends downstream to the wing trailing edge for this particular value of α.



(a) Contours of Mach number (b) C_f and C_P distributions over the wing surface

Figure 3. Steady 2.5D RANS solution (baseflow) for OAT15A swept wing at $\alpha = 2.95°$, $M_n = 0.73$

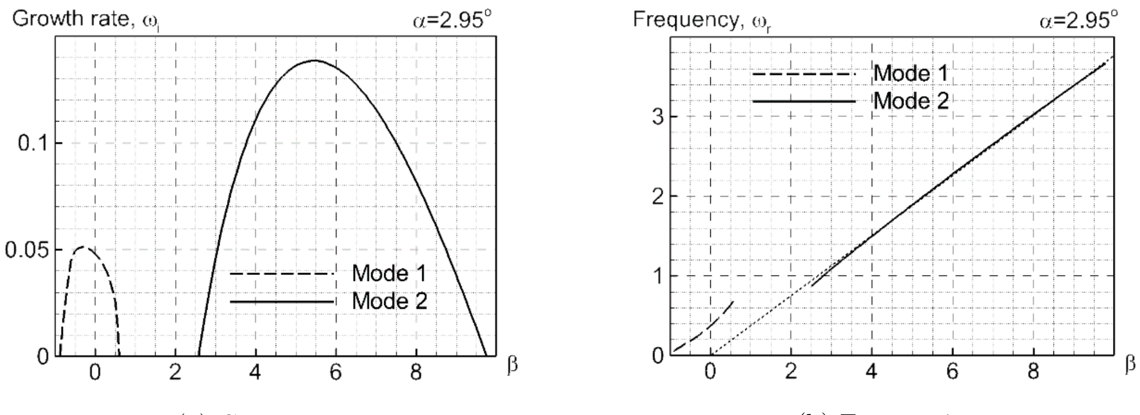
As shown in [2, 14], for transonic flows past swept wings, the q-3D analysis predicts two modes of instability, an oscillatory mode and a traveling propagating outboard mode. There can be two characteristic wavelengths for the traveling modes ($\lambda_z \approx c$ and/or $\lambda_z \approx 0.1c$), depending on the values of M_n and α. Figure 4 illustrates the mode shapes for the u-component of the eigenvectors corresponding to the spanwise uniform ($\beta = 0$) oscillatory mode and the traveling mode at $\beta = 2\pi$ ($\lambda_z = c$). The Figure shows that the two modes are qualitatively similar. The oscillatory mode has maximums at the shock and in the separated shear layer, with the magnitude at the shock roughly 20 times larger than in the shear layer. In contrast to this, for the traveling mode, the magnitudes of u-component of the eigenvector at the shock and in the separated shear layer differ by only a factor of 5.



(a) Oscillatory ($\beta = 0$) mode (b) Traveling ($\beta = 2\pi$, $\lambda_z = c$) mode

Figure 4. Contours of magnitude of u-component of eigenvectors of instability, normalized by the maximum, from q-3D stability analysis at $\alpha = 2.95°$, $M_n = 0.73$

Figure 5 presents quantitative characteristics for the two unstable modes, showing the growth-rates and the frequencies of these modes as functions of the wavenumber β for the baseflow shown in Fig. 3 (negative values of β in this figure correspond to waves propagating in the direction opposite to the direction of the z axis, i.e. inboard).



(a) Growth rates (b) Frequencies

Figure 5. Characteristics of the oscillatory (Mode 1) and travelling (Mode 2) unstable modes as a function of wavenumber from q-3D stability analysis at $\alpha = 2.95°$, $M_n = 0.73$

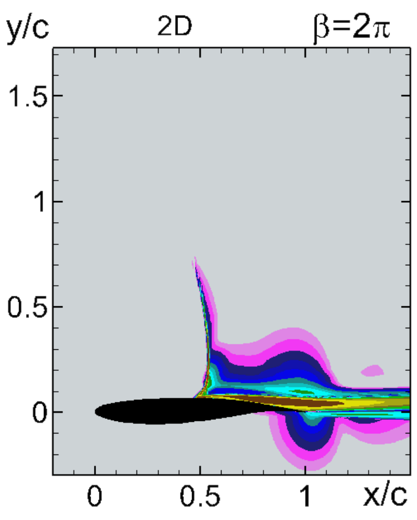
The Figure shows that the oscillatory mode (Mode 1 in the figure legend) is observed for $|\beta| < 1$, and is characterized by long-wavelength disturbances ($\lambda_z/c > 2\pi$). The frequency of these disturbances ω_r is less than ≈ 0.7 ($St = fc/U_n \approx 0.11$) and the maximum growth rate $\omega_{i,max}$ is equal to ≈ 0.05. The maximum growth is reached in the vicinity of $\beta = 0$ (i.e., for the disturbances nearly uniform in the z-directions), and shows a slight bias towards inboard propagation.

The unstable traveling mode, which results in a spanwise undulating shape of the shock [9], is observed at wavenumbers β in the range from ≈ 2.5 up to ≈ 10 (λ_z/c from ≈ 0.63 up to ≈ 2.5). These wavelengths are intermediate to the long-wavelength oscillatory mode and shorter-wavelength traveling disturbances ($\lambda_z/c \approx 0.1$) observed for some flow conditions [2, 3]. The traveling mode frequency increases linearly with β and is significantly higher than that of the oscillatory mode. The maximum growth rate of the traveling mode occurs at $\beta \approx 5.5$ and is equal to ≈ 0.14 (almost 3 times larger than that of the oscillatory mode). For this reason, this mode has been observed to be dominate for both infinite [9] and finite [11, 23] swept wings. The following section focusses on this traveling mode to assess the different approaches used for the prediction of buffet onset.

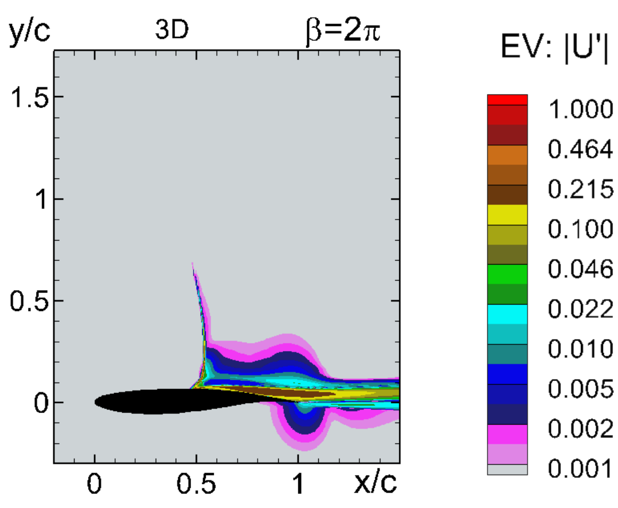
3.2. Comparison of Approaches for Buffet Onset Prediction

The comparison of stability characteristics of the traveling mode predicted by q-3D analysis and fully-3D analysis at $\alpha = 2.95°$ is presented in Fig. 6 and 7.

Figure 6 compares the predicted mode shapes in the form of the magnitude of the u-component of velocity at $\beta = 2\pi$. In the q-3D approach, this quantity is part of the 2D eigenfunction, and in the fully-3D analysis it is derived by spanwise-averaging of the 3D eigenfunction. Figure 7 presents the growth rate and frequency of the traveling mode as functions of β. The Figures show that the q-3D and fully-3D analyses return qualitatively-similar results for both the shape and the growth rate and frequency.

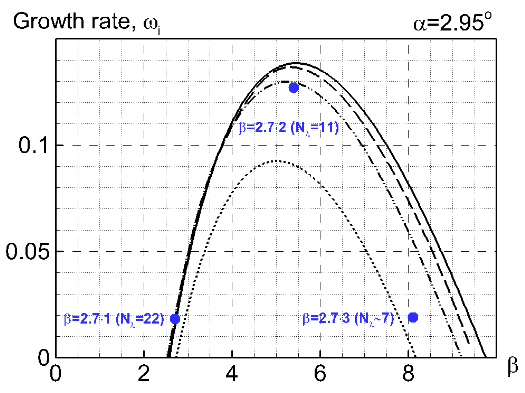


(a) Q-3D stability analysis

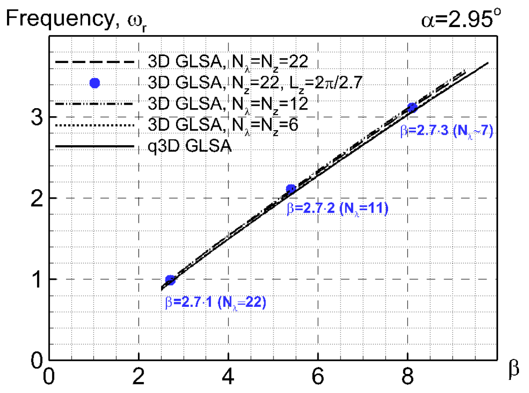


(b) Fully-3D stability analysis

Figure 6. Contours of magnitude of u-component of traveling-mode eigenvectors normalized by the maximum at $\beta = 2\pi$ ($\lambda_z = c$) and $\alpha = 2.95°$, $M_n = 0.73$



(a) Growth rates



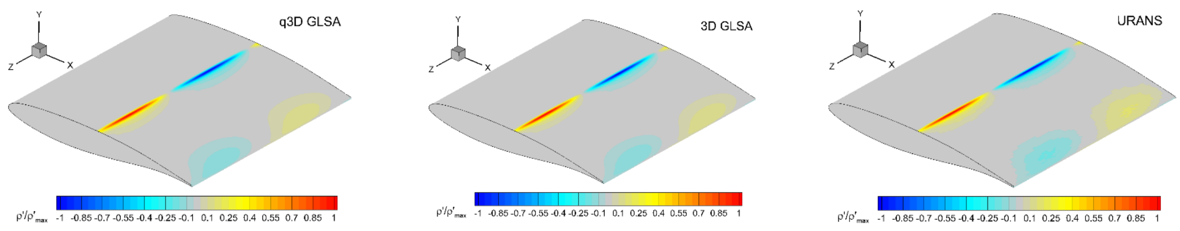
(b) Frequencies

Figure 7. Comparison of characteristics for travelling mode of instability from q-3D and fully-3D stability analysis as functions of wavenumber at $\alpha = 2.95°$, $M_n = 0.73$. GLSA stands for Global Linear Stability Analysis. Circles show the three least stable eigenvalues calculated at $\beta = 2.7$, $N_z = 22$

The growth rate for the fully-3D analysis is dependent on the number of grid planes in the z direction, but converges to the q-3D result with increasing grid resolution N_z. The fully-3D results (shown by the dashed lines) are based on $\beta = 2\pi/L_z$, so the N_z value is the number of grid planes per wavelength λ_z. As the value of N_z is increased from 6, to 12, and to 22, the fully-3D growth rates approach the q-3D curve. As noted earlier (and discussed in [8]), the fully-3D solution includes a set of discrete modes with $\lambda_z = L_z/m$, or $\beta = m \cdot 2\pi/L_z$. An example of these modes is shown by the blue symbols, for $\beta = m \cdot 2.7$, and $m = 1, 2, 3$. At the wavelength corresponding to $m = 1$, the N_z per wavelength is 22, and the growth rate falls on the curve associated with $N_z = 22$. For $m = 2$, the N_z per wavelength is 11, and the growth rate is just below the curve for $N_z = 12$. Likewise, for $m = 3$, the N_z per wavelength is approximately 7, and the growth rate is close to the $N_z = 6$ curve. These results show that the fully-3D growth rate is very sensitive to the N_z per wavelength, so that unless the $m = 1$ mode is highly over-resolved, the higher modes ($m > 1$) need to be viewed with caution.

The frequencies shown in Fig. 7 have a much weaker sensitivity to the spanwise grid resolution. These frequencies are set by the spanwise propagation of the fixed-wavelength mode (along with the sweep angle [2, 14]), so they are not strongly dependent on finer details of the mode shape.

Finally, a direct comparison of the shapes of the 3D unstable traveling mode predicted by the q-3D and fully-3D stability analyses and by the 3D URANS simulations is provided in Fig. 8. This shows instantaneous fields of the normalized amplitude of the density component of this mode at $\beta = 2\pi$ ($\lambda_z/c = 1$) and $\alpha = 2.95°$ on the wing upper surface computed with the use of these three approaches. This comparison shows very good agreement for the spatial structure of the unstable modes.

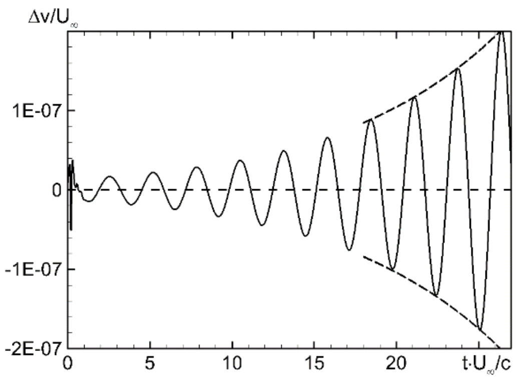


(a) Q-3D stability analyses (b) Fully-3D stability analyses (c) 3D URANS simulation

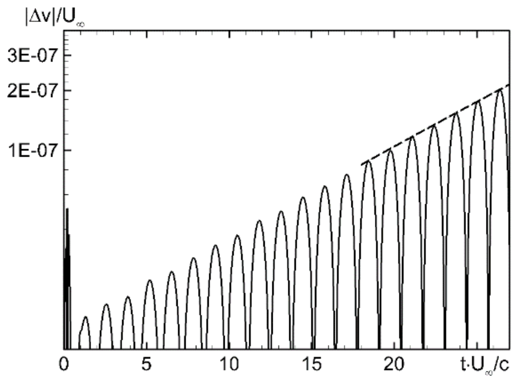
Figure 8. Comparison of normalized amplitude of density component of eigenvector on the airfoil surface at $\beta = 2\pi$ ($\lambda_z/c = 1$) and $\alpha = 2.95°$, $M_n = 0.73$

We now consider the temporal variation of the 3D URANS solutions for the spanwise domain $L_z = 2c$ started from the steady 2.5D RANS solutions at supercritical values of the angle of attack. In general, after some transient period, the flow becomes unsteady with a periodic spatial variation in the spanwise direction with wavelength $\lambda_z/c = 1$ ($\beta = 2\pi$). This wavelength is close to the wavelength with the maximum growth rate for the traveling unstable mode, according to both the fully-3D and q-3D stability analysis (see Fig. 6a) and is admissible by the spanwise domain length $L_z = 2c$. At this stage of development, the amplitudes of the oscillations for all aerodynamic variables (i.e. their maximum deviation from the initial conditions) grow exponentially in time. This growth is illustrated in Fig. 9, showing the time-evolution of the v-component of the unsteady disturbance at the point $x = 1.05$, $y = 0$ for $\alpha = 2.95°$, as an example. The growth rate ω_i deduced from these curves is equal ≈ 0.1, which is roughly 0.03 below the stability results.

Similar post-processing of the 3D URANS solutions at other values of the angle of attack in the range 2.8° up to 3.0°, gives the dependence of $\omega_i(\alpha)$ shown in Fig. 10 together with the similar dependencies based on the q-3D and fully-3D stability analysis. The very small shift in the growth rate for the fully-3D stability analysis, compared to the q-3D result, is linked to the spanwise grid resolution as discussed for Fig. 7. The slightly lower growth rates for the URANS results are attributed to differences in the discretization errors compared to the stability formulations. Recall that the stability equations are derived and discretized independently, as opposed to using the discrete linearized operator from the URANS. The observation of generally-lower growth rates in the URANS for near-critical conditions has been observed in other applications as well [4, 5].



(a) Linear y-axis



(b) Exponential y-axis

Figure 9. 3D URANS simulation results for the time-evolution of the v-velocity component of the disturbance in the linear stage of solution at $\alpha = 2.95°$, $M_n = 0.73$. Dashed lines show amplitude of the disturbances as a function of time

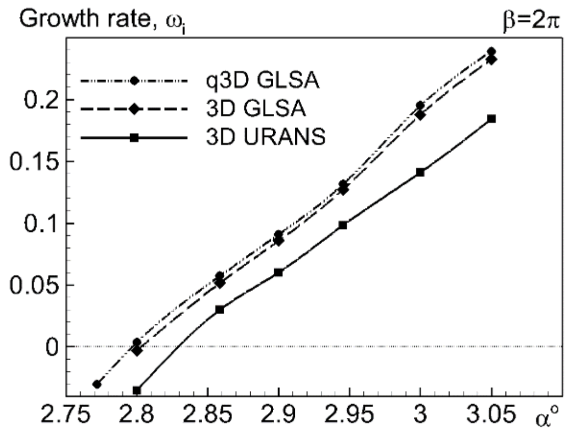


Figure 10. Comparison of growth rates for the travelling mode of instability at $\beta = 2\pi$ as a function of angle of attack, from q-3D and fully-3D stability analysis and 3D URANS simulations, $M_n = 0.73$

The small shifts in the growth rates translate to differences in critical conditions for the onset of buffeting as presented in Tab. 1. The value of α_{crit} is somewhat larger for the URANS simulations compared to the stability analysis. However, the difference of α_{crit} does not exceed 0.03 degrees (roughly 0.15% difference in lift at buffet onset). The maximum difference of the frequency of the disturbances predicted by the three approaches is about 2%. In practice, these overall differences are considered to be very small compared to other potential contributors (e.g. turbulence-model parameters).

Table 1. Critical angles of attack and corresponding frequencies for travelling mode at $\beta = 2\pi$ predicted by 3D URANS and the q-3D and fully-3D stability analysis for $M_n = 0.73$

Approach	α_{crit}	ω_r
3D URANS	2.83	2.36
3D stability	2.81	2.43
q-3D stability	2.80	2.37

Conclusion

A comparative study is performed for the three URANS-based approaches to predicting buffet onset and its characteristics for an infinite swept wing. Results are presented for the supercritical OAT15A airfoil, with that as an example. The most general of these approaches is based on the direct numerical solutions of 3D URANS equations, while two other approaches employ fully-3D and quasi-3D linear global stability analysis of the corresponding steady 2.5D RANS solutions. For this spanwise-uniform flow, q-3D stability analysis provides the most efficient and accurate representation of the instabilities leading to buffet onset. The fully-3D stability analysis, which does not require spanwise uniform flow, provides a potential link to more general wing configurations similar to the 3D URANS simulations.

Based on the analysis of the results of the 3D URANS and comparisons to the q-3D and fully-3D stability analysis, both stability formulations are shown to accurately capture the initial disturbance development. Spanwise-wavelength selection for the fully-3D analysis, as for the 3D URANS simulations, is dependent on the spanwise domain length. By varying the domain length, the fully-3D stability results are shown to capture the growth rates and frequencies predicted by the q-3D analysis. In addition, the spatial distributions of the disturbances over the wing cross-section plane are shown to be in good agreement for the different approaches. Stability predictions for the critical angle of attack for buffet onset at a given Mach number are in good agreement with the 3D URANS simulations (the difference is within the range of 0.03°, representing less than 0.15% in lift at buffet onset). The good overall agreement for these three independent formulations applied to the infinite swept wing provides a cross validation of the approaches.

Acknowledgements

The study is conducted with financial support of Russian Scientific Foundation, Grant No. 22-11-00041. The computations were performed on the HP computing facilities of the Peter the Great Saint-Petersburg Polytechnic University (http://www.spbstu.ru).

References

1. Crouch, J.D., Garbaruk, A., Magidov, D., Travin, A.: Origin of transonic buffet on aerofoils. Journal of Fluid Mechanics 628, 357–369 (2009). `https://doi.org/10.1017/S0022112009006673`

2. Crouch, J.D., Garbaruk, A., Strelets, M.: Global instability in the onset of transonic-wing buffet. Journal of Fluid Mechanics 881, 3–22 (2019). `https://doi.org/10.1017/jfm.2019.748`

3. Crouch, J.D., Garbaruk, A., Strelets, M.: Global instability in the onset of transonic-wing buffet – CORRIGENDUM. Journal of Fluid Mechanics 901, E1 (2020). `https://doi.org/10.1017/jfm.2020.557`

4. Crouch, J., Garbaruk, A., Magidov, D.: Predicting the onset of flow unsteadiness based on global instability. Journal of Computational Physics 224(2), 924–940 (2007). https://doi.org/10.1016/j.jcp.2006.10.035

5. Garbaruk, A., Crouch, J.D.: Quasi-three dimensional analysis of global instabilities: onset of vortex shedding behind a wavy cylinder. Journal of Fluid Mechanics 677, 572–588 (2011). https://doi.org/10.1017/jfm.2011.102

6. Giannelis, N.F., Vio, G.A., Levinski, O.: A review of recent developments in the understanding of transonic shock buffet. Progress in Aerospace Sciences 92, 39–84 (2017). https://doi.org/10.1016/j.paerosci.2017.05.004

7. Gorobets, A.V., Duben, A.P.: Technology for supercomputer simulation of turbulent flows in the good new days of exascale computing. Supercomputing Frontiers and Innovations 8(4), 4–10 (Feb 2022). https://doi.org/10.14529/jsfi210401

8. He, W., Timme, S.: Triglobal infinite-wing shock-buffet study. Journal of Fluid Mechanics 925, A27 (2021). https://doi.org/10.1017/jfm.2021.678

9. Iovnovich, M., Raveh, D.E.: Numerical study of shock buffet on three-dimensional wings. AIAA Journal 53(2), 449–463 (2015). https://doi.org/10.2514/1.J053201

10. Jacquin, L., Molton, P., Deck, S., *et al.*: Experimental study of shock oscillation over a transonic supercritical profile. AIAA Journal 47(9), 1985–1994 (2009). https://doi.org/10.2514/1.30190

11. Koike, S., Ueno, M., Nakakita, K., Hashimoto, A.: Unsteady pressure measurement of transonic buffet on NASA common research model. In: 34th AIAA Applied Aerodynamics Conference, Washington, D.C., USA, June 13–17, 2016. https://doi.org/10.2514/6.2016-4044

12. Masini, L., Timme, S., Peace, A.J.: Analysis of a civil aircraft wing transonic shock buffet experiment. Journal of Fluid Mechanics 884, A1 (2020). https://doi.org/10.1017/jfm.2019.906

13. Paladini, E., Marquet, O., Sipp, D., *et al.*: Various approaches to determine active regions in an unstable global mode: application to transonic buffet. Journal of Fluid Mechanics 881, 617–647 (2019). https://doi.org/10.1017/jfm.2019.761

14. Paladini, E., Beneddine, S., Dandois, J., *et al.*: Transonic buffet instability: From two-dimensional airfoils to three-dimensional swept wings. Phys. Rev. Fluids 4, 103906 (Oct 2019). https://doi.org/10.1103/PhysRevFluids.4.103906

15. Plante, F., Dandois, J., Beneddine, S., *et al.*: Link between subsonic stall and transonic buffet on swept and unswept wings: from global stability analysis to nonlinear dynamics. Journal of Fluid Mechanics 908, A16 (2021). https://doi.org/10.1017/jfm.2020.848

16. Poplingher, L., Raveh, D.E., Dowell, E.H.: Modal analysis of transonic shock buffet on 2D airfoil. AIAA Journal 57(7), 2851–2866 (2019). https://doi.org/10.2514/1.J057893

17. Sartor, F., Timme, S.: Delayed detached-eddy simulation of shock buffet on half wing-body configuration. AIAA Journal 55(4), 1230–1240 (2015). https://doi.org/10.2514/1.J055186

18. Shur, M., Strelets, M., Travin, A.: High-order implicit multi-block Navier–Stokes code: Ten-year experience of application to RANS/DES/LES/DNS of turbulence. https://cfd.spbstu.ru/agarbaruk/doc/NTS_code.pdf (2008), accessed: 2009-10-01

19. Spalart, P., Allmaras, S.: A one-equation turbulence model for aerodynamic flows. In: 30th Aerospace Sciences Meeting and Exhibit, Reno, NV, USA, January 6–9, 1992. https://doi.org/10.2514/6.1992-439

20. Spalart, P.: Trends in turbulence treatments. In: Fluids 2000 Conference and Exhibit, Denver, CO, USA, June 19–22, 2000. https://doi.org/10.2514/6.2000-2306

21. Sugioka, Y., Koike, S., Nakakita, K., *et al.*: Experimental analysis of transonic buffet on a 3D swept wing using fast-response pressure-sensitive paint. Experiments in Fluids 59(6) (jun 2018). https://doi.org/10.1007/s00348-018-2565-5

22. Theofilis, V.: Global linear instability. Annual Review of Fluid Mechanics 43(1), 319–352 (2011). https://doi.org/10.1146/annurev-fluid-122109-160705

23. Timme, S.: Global instability of wing shock-buffet onset. Journal of Fluid Mechanics 885, A37 (2020). https://doi.org/10.1017/jfm.2019.1001

DOI: 10.14529/jsfi220402

One Case of Shock-free Deceleration of a Supersonic Flow in a Constant Cross Section Area Channel

Dmitry E. Khazov[1]

Air flows with supersonic speeds are used in many cases, for example, as aircraft air intakes, wind tunnels, and energy separation devices. In many cases it is necessary to decelerate the flow to sonic speeds. Traditionally the deceleration realized through the shocks system, which leads to total pressure losses. The article considers the method of deceleration of supersonic flows using permeable surfaces. In this case, the deceleration process occurs without shocks and, therefore, with lower total pressure losses. We have considered the flow in a tube with permeable wall located behind a supersonic nozzle. One-dimensional and axisymmetric mathematical models of such a device are developed. The calculation results are compared with experimental data. It is shown that, depending on the ratio of pressure inside the tube and the ambient pressure, different flow regimes inside the tube are possible: pure subsonic, transitional from supersonic to subsonic, and pure supersonic. The transition from supersonic to subsonic flow occurs without shocks due to the suction and friction combined effects.

Keywords: supersonic flow, permeable wall, Mach number, deceleration, injection/suction.

Introduction

The problem of a viscous supersonic flow deceleration in channels is of interest to researchers and engineers due to the importance of this problem for modern and future jet engines and wind tunnels. In the case of a constant cross section channel with impermeable walls, the supersonic flow is decelerated through a complex shocks structure and near-wall separation regions, generally called a pseudo-shock [4]. The deceleration of supersonic flow in the shock waves leads to the additional total pressure losses. Considerable potential in this regard has the use of permeable surfaces.

In the work [3], permeable (perforated) boundaries were used to accelerate the flow from sonic to supersonic speeds, as well as to equalize the non-uniformity of the supersonic flow. The authors of the paper [11] performed an experimental study of the flow in a permeable tube of constant cross section, installed in a supersonic nozzle. Experiments have shown that a transition from subsonic to supersonic flow occurs inside the tube.

As it can be seen from the works cited above, the use of permeable surfaces demonstrates the possibility of supersonic flow control. The purpose of this work is to study the processes of a supersonic flow deceleration in a channel of constant cross section with a permeable wall. To achieve this goal, it is necessary to develop numerical models that describe supersonic flows (with the possibility of passing through the critical point) in a channel of constant cross section with a permeable wall. Validate these models on the available experimental data, as well as perform a parametric study.

The article is organized as follows. Section 1 is devoted to the problem statement. In Section 2, we described two numerical models. The Section 3 discusses the main results of this study. The Conclusion summarizes the results of the study and indicates directions for further work.

[1]Institute of Mechanics, Lomonosov Moscow State University, Moscow, Russian Federation

1. Problem Statement

Let us consider the flow of a perfect gas in a channel of constant cross section with mass suction and friction. This type of flow can be found in transpiration cooling devices, fuel supply channels in burners, etc. In general, the device consists of a nozzle and a working channel with permeable wall. The permeable wall can be made of perforated or porous, permeable material. The parameters of porous materials, such as porosity and particle size, provide a different surface mass flux for the same pressure drop.

This article considers a device consisting of a supersonic nozzle and a cylindrical channel of constant cross section with a permeable wall (see Fig. 1). The gas accelerates to supersonic speeds in the nozzle and then enters the channel, where the flow decelerated. Since the wall is a permeable the pressure ratio in the channel p_{inn} and the ambient pressure p_{amb} will determine the direction of the flow through the wall: by $p_{inn} > p_{amb}$ gas will be sucked out from the channel; by $p_{inn} < p_{amb}$ gas will be injected into the channel. Thus, depending on the pressure drop inside the channel and the ambient pressure, various flow regimes can be realized.

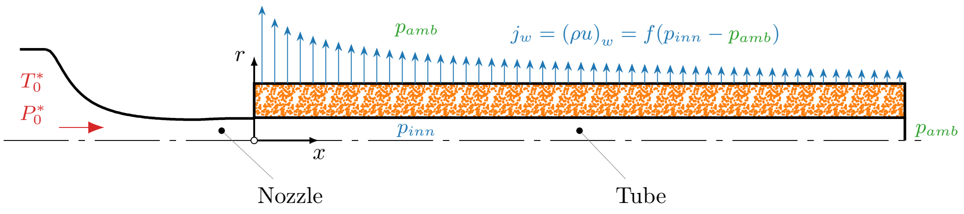


Figure 1. A sketch of the considered device

2. Numerical Models

We used one- and two-dimensional (axisymmetric) mathematical models for a detailed study of the processes occurring in the channel with permeable wall. The one-dimensional model allows to obtain the distribution of the main parameters (velocity, pressure, temperature, etc.) along the channel. It is assumed that all parameters are uniformly distributed across the section. In turn, the two-dimensional model allows to get more detailed information about the processes taking place inside the device. However, the use of such models is much more time consuming, as for the stage of model creation, and at the stage of a solution.

2.1. One-Dimensional Model

We used the well-known Shapiro–Hawthorne method [1] for a one-dimensional gas flow model. The main idea of the method is that the differential of each variable (velocity, pressure, temperature, etc.) is expressed through a linear combination of independent elementary factors of influence (such as friction, area change, external heat exchange, etc.); the coefficients of these linear combinations, called “influence coefficients”, are expressed as functions of one variable (Mach number).

Based on the balance relations for the selected elementary volume and using the equation of state (perfect gas), we can obtain the following equation for the Mach number changing in a constant area channel with friction, heat and mass transfer:

$$\frac{d\mathrm{M}^2}{\mathrm{M}^2} = \overbrace{\frac{1+k\mathrm{M}^2}{1-\mathrm{M}^2}\frac{dQ_w}{mc_pT}}^{\text{heat transfer}} + \overbrace{\frac{k\mathrm{M}^2\left(1+\frac{k-1}{2}\mathrm{M}^2\right)}{1-\mathrm{M}^2}4c_f\frac{dx}{d_h}}^{\text{friction}} + \overbrace{\frac{2\left(1+k\mathrm{M}^2\right)\left(1+\frac{k-1}{2}\mathrm{M}^2\right)}{1-\mathrm{M}^2}\frac{dm}{m}}^{\text{injection/suction}}, \tag{1}$$

where m is the mass flow rate, c_p is the specific heat at constant pressure, T is the thermodynamic temperature, Q_w is the net heat flow, c_f is the friction coefficient and d_h is the hydraulic diameter of the channel.

Equations for other variables (pressure, temperature, etc.) can be found in [1].

The amount of the heat (removed or added to the main flow) was determined from the following relation:

$$dQ_w = 4q_w\frac{A}{d_h}dx, \quad q_w = j_w c_p\left(T_{aw} - T^*\right), \tag{2}$$

where A is the cross-sectional area, $j_w = (\rho u)_w$ is the mass flux through permeable wall, T_{aw} is the adiabatic wall temperature and T^* is the stagnation temperature of the flow.

The value of mass flux j_w was determined from the Darcy–Forchhämer law for a cylindrical tube with inner and outer diameters d_{inn} and d_{out}, respectively [7]:

$$\frac{p_{amb}^2 - p_{inn}^2}{\mathrm{R}T\Delta d} = \alpha\mu\frac{d_{inn}}{\Delta d}\ln\frac{d_{out}}{d_{inn}}j_w + \beta\frac{d_{inn}}{d_{out}}j_w^2, \tag{3}$$

where p_{inn} is the pressure at the inner surface of a wall, $\Delta d = d_{out} - d_{inn}$ is the diameter difference, μ is the molecular viscosity and R is the specific gas constant.

Let us assume that a porous tube consists of uniform spherical particles of diameter d_p. In this case, values of viscous α and inertial β coefficients can be obtained from [7]:

$$\alpha = \frac{171\left(1-\varepsilon\right)^2}{\varepsilon^3 d_p^2}, \qquad \beta = \frac{0.635\left(1-\varepsilon\right)}{\varepsilon^{4.72}d_p}. \tag{4}$$

Values of porosity ε and diameter of spherical particles d_p determined based on the experimental flow characteristic of the sample of a permeable wall given in [12] which was used in present experiments and were as follows:

$$\varepsilon \approx 34\% \qquad d_p = 70\times10^{-6}\ \mathrm{m}. \tag{5}$$

The friction coefficient was determined from the Colebrook–White ratio [9]:

$$\frac{1}{\sqrt{\lambda}} = -2\log_{10}\left(\frac{2.51}{\mathrm{Re}\sqrt{\lambda}} + \frac{\Delta_s}{3.7}\right), \quad c_{f_0} = \frac{\lambda}{4}, \tag{6}$$

where λ is the Darcy friction factor, $\Delta_s = h_s/d_h$ is the relative roughness. The Reynolds number is defined as $\mathrm{Re} = \rho u d_h/\mu$ with ρ the fluid density, d_h the hydraulic channel diameter, u the mass-mean fluid velocity and μ the fluid viscosity.

There is a need to take into account the effects of compressibility Ψ_M and injection/suction Ψ_b on the friction coefficient according to [6]:

$$c_f = \Psi_\Sigma c_{f_0}, \quad \Psi_\Sigma = \Psi_\mathrm{M}\Psi_b, \tag{7}$$

$$\Psi_\mathrm{M} = \left(\frac{\arctan \mathrm{M}\sqrt{r\frac{k-1}{2}}}{\mathrm{M}\sqrt{r\frac{k-1}{2}}}\right)^2, \quad \Psi_b = \left(1 - \frac{b}{b_{cr}}\right)^2, \quad b_{cr} = 4, \tag{8}$$

where permeability parameter b was obtained by following:

$$b = \frac{\bar{j}_w}{c_{f0}/2}, \quad \bar{j}_w = \frac{j_w}{(\rho u)_\infty}. \tag{9}$$

Massflow change was obtained from following relation:

$$dm = j_w d_h \pi dx. \tag{10}$$

Thus, using Eq. (1) (and others for pressure, temperature, etc.) and the closing relations Eqs. (2)–(10), it is possible to form a closed system of equations describing the flow in a channel with friction, heat transfer and injection/suction through a permeable wall. The system can be numerically integrated under the appropriate initial conditions:

$$\mathrm{M} = \mathrm{M}_0,\ u = u_0,\ T = T_0,\ p = p_0 \quad \text{at} \quad x = 0. \tag{11}$$

2.2. Two-Dimensional Model

The problem was modeled in the axisymmetrical formulation by using of ANSYS Fluent. Structured mesh was created by gmsh [2] preprocessor. The mesh size was about 10^5 cells. The discretization of the Reynolds-averaged Navier–Stokes equations (RANS), the energy equations and the equations of the corresponding turbulence model was performed on the basis of the control volume method. The second order upwind scheme was used for the spatial discretization. Based on previous study [5] the standard $k-\omega$ turbulence model was used.

The permeable wall was not modeled explicitly. The mass flux and heat flux were applied at the internal cylindrical surface of the porous tube. The value of mass flux was obtained on the basis of the Darcy–Forchhämer equation (3).

The following boundary conditions were used for the 2D model. Total pressure P_0^* and temperature T_0^* were specified at the inlet (see Fig. 1). The static pressure (p_{amb}) was specified at the outlet. The value of the specified static pressure is used only while the flow is subsonic. For the supersonic flow the pressure will be extrapolated from the flow in the interior.

3. Results and Discussion

We used two mathematical models developed above to simulate the flow in the experimentally investigated device in [8]. The profiled axisymmetric supersonic nozzle was used. The divergent section was profiled by using the method of characteristics. The nominal Mach number determined from the area ratio of the throat ($d_{cr} = 3.2$ mm) and the exit section ($d_{ex} = 3.4$ mm) in the case of isentropic air expansion is $\mathrm{M}_{is} = 1.43$. The coordinates of the nozzle contour are given in [8]. The permeable tube was made of synthetic corundum, $L = 150$ mm in length, $d_{out} = 10.4$ mm in the outer diameter, $d_{inn} = 3.5$ mm in the inner diameter. Note that the experimentally investigated permeable tube's inner surface was treated as rough with $h_s = d_p/2 = 35 \times 10^{-6}$ m ($\Delta_s = 0.01$).

The inlet total temperature was equal $T_0^* = 295.6$ K for all cases. The ambient pressure was equal $p_{amb} = 0.1$ MPa.

We extracted mass mean values from the 2D model for comparison with experimental and 1D data. Then we provided the parametric study by using the validated 2D model.

Figure 2 shows distribution of main flow parameters along the channel by $P_0^* = 0.4$ MPa. In addition to measured data (symbols), calculation data (lines) are also shown. The correspondence

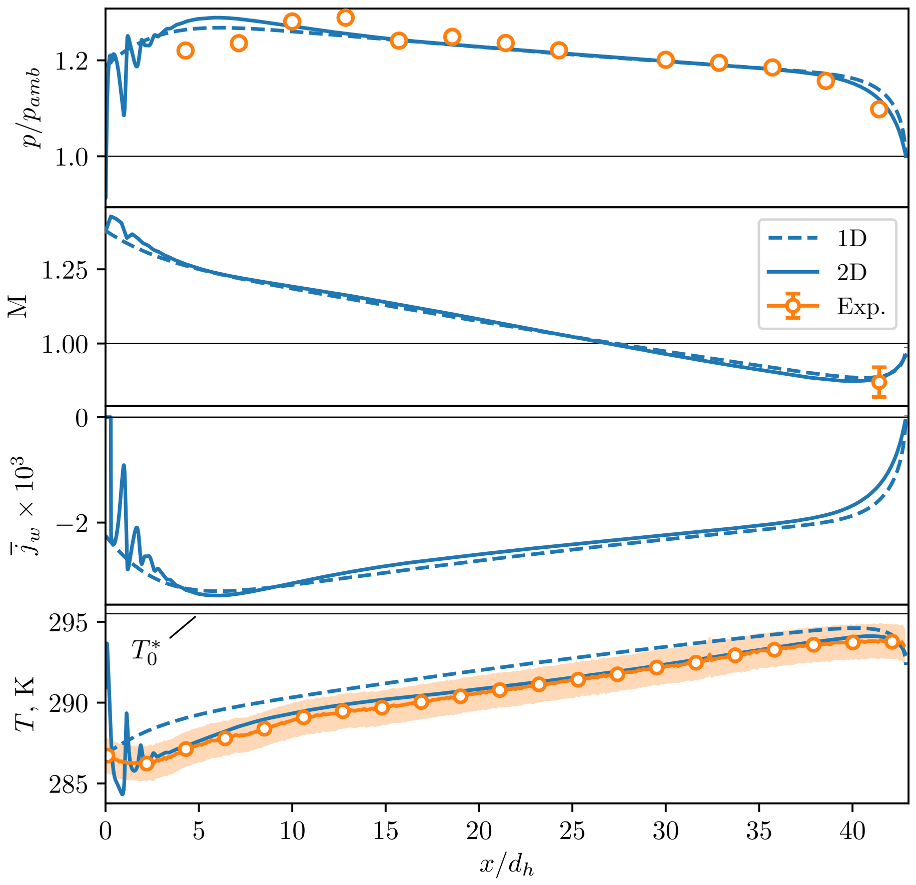


Figure 2. Main parameters distribution along the channel with a permeable wall. $P_0^* = 0.4$ MPa

between measured and calculated data is acceptable. The difference in temperature distribution is observed at the beginning of the channel ($x/d_h < 5$). It can be explained by the presence of shock waves generated by the backward step between nozzle and tube (see Fig. 3). As mentioned above, in the numerical model, the porous tube was not modeled explicitly. Therefore, the peaks of the wall temperature were observed. In contrast, in the experiment, the influence of shock waves on the wall temperature, as we assume, was smoothed by the porous tube thermal conductivity.

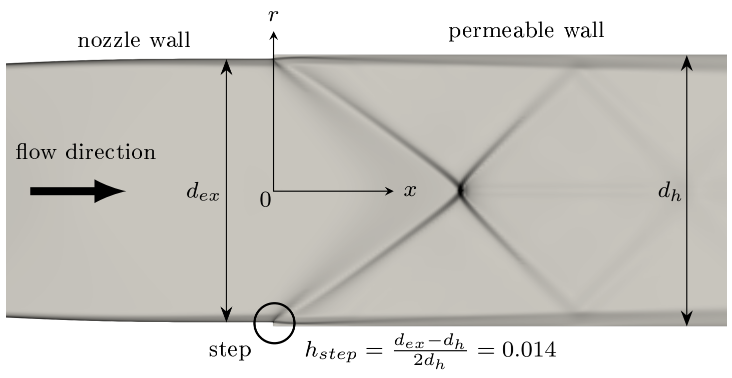


Figure 3. Numerical Schlieren of flow in the channel with a permeable wall. $P_0^* = 0.4$ MPa

The radial stagnation pressure distributions and static pressure at the wall were measured at $x/d_h = 41.4$ to obtain the Mach number distribution. Results of recovered Mach number distributions are shown in Fig. 4. In addition to the measured values (symbols), the figure also shows the calculated data (lines). Mach number distribution along the channel with permeable wall is shown in Fig. 5.

From the Fig. 4 we can see that at lower initial stagnation pressures ($P_0^* < 0.5$ MPa) the flow is sub- or transonic at the section $x/d_h = 41.4$, although, for these pressures, the velocity at the nozzle exit is supersonic (see Fig. 5). This circumstance requires clarification.

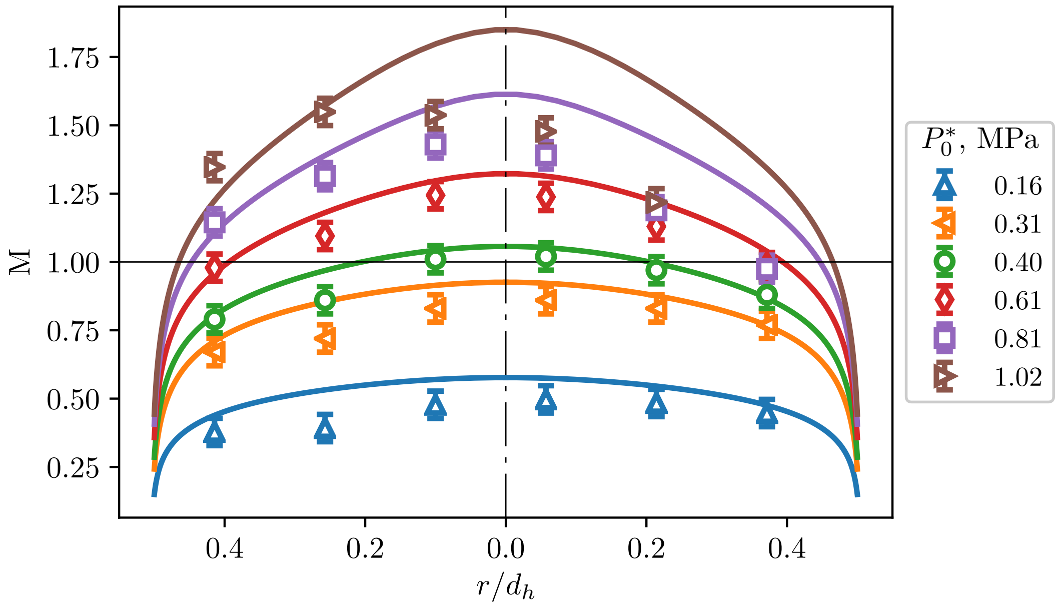


Figure 4. Mach number radial distribution at $x/d_h = 41.4$ by different initial stagnation pressure in plenum. Symbols are measured data; solid lines are results of 2D calculations

As it can be seen from the Fig. 2, throughout the entire length of the channel, the pressure exceeds atmospheric and, therefore, gas is sucked out along the whole length of the channel ($\bar{j}_w < 0$) through the permeable wall. The combination of the suction and friction leads to the fact that at a certain length ($x/d_h \approx 27$), the Mach number takes a critical value M = 1, and then the Mach number goes into the subsonic flow region. It should be noted that the transition, according to the calculation results, is not accompanied by shock waves, which can also be judged by the measured pressure distribution (see Fig. 2). As the pressure in the plenum increases, the critical section inside the channel (M = 1) shifts to the outlet section (see Fig. 5) and, starting from some value of $P_0^* > 0.5$ MPa, the flow throughout the whole length of the channel remains supersonic. The possibility of such passage through the sonic point will be discussed below.

Let us consider the physical possibility of the shock-free deceleration of a supersonic flow using a one-dimensional model. The Eq. (1) can be rewritten as it has been done in [10]:

$$\frac{1-\mathrm{M}^2}{\mathrm{M}^2}\frac{d\mathrm{M}^2}{dx} = G(x), \tag{12}$$

where

$$\begin{aligned} G(x) = \frac{1+k\mathrm{M}^2}{mc_pT}\frac{dQ_w}{dx} &+ k\mathrm{M}^2\left(1+\frac{k-1}{2}\mathrm{M}^2\right)\frac{4c_f}{d_h} + \\ &+ 2\left(1+k\mathrm{M}^2\right)\left(1+\frac{k-1}{2}\mathrm{M}^2\right)\frac{1}{m}\frac{dm}{dx} = \\ &= G_Q + G_f + G_m, \end{aligned} \tag{13}$$

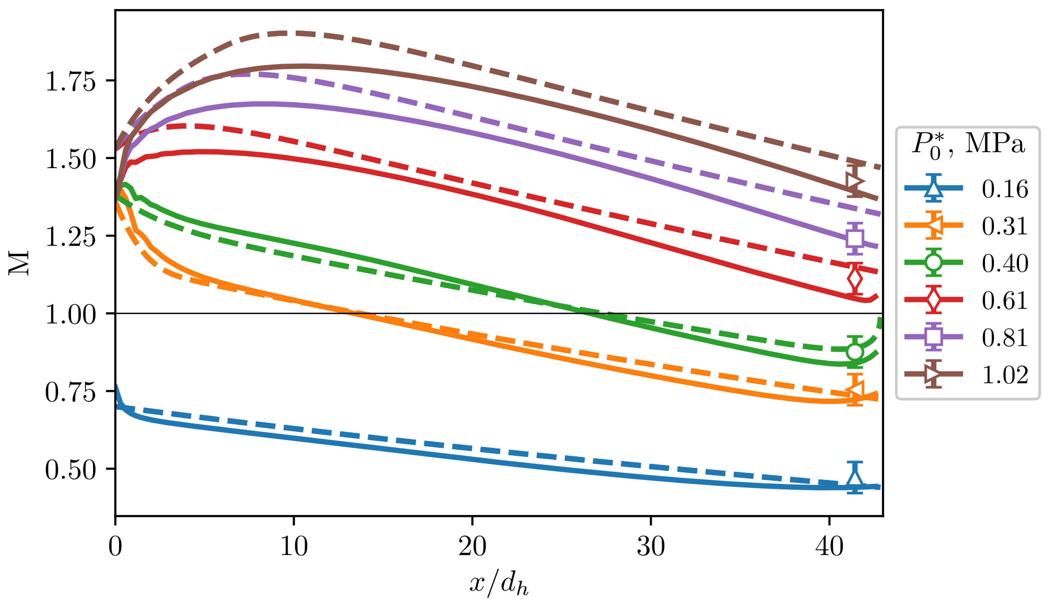


Figure 5. Mach number distribution along the channel with a permeable wall by different initial stagnation pressure in the plenum. Symbols are measured data; dashed lines are results of 1D calculations; solid lines are 2D calculations

where G_Q, G_f and G_m are elementary actions induced by heat transfer, friction and injection/suction, correspondingly.

Equation (12) shows that the local Mach number increases or decreases along the channel depending on the flow regime (subsonic or supersonic), as well as on whether the function G (summary action) is positive or negative, according to Tab. 1.

Table 1. Relations between G and $d\mathrm{M}^2/dx$ [10]

	$\mathrm{M} < 1$	$\mathrm{M} = 1$	$\mathrm{M} > 1$
$G < 0$	$-$	∞	$+$
$G = 0$	0	0/0	0
$G > 0$	$+$	∞	$-$

Therefore, the Mach number along the channel can vary in different ways, depending on whether the initial Mach number (M_0 at $x = 0$) is less or greater than one. Depending on whether the function G is always positive, negative, or changes sign. Figure 6 shows all the possible options.

As it is shown in the Fig. 6, the G function should change its value from positive to negative to implement the supersonic flow's deceleration to subsonic speeds (see Fig. 6e).

Figure 7 shows the changing of the Mach number and components of the function (see Eq. (13)) $\overline{G} = G/G(0)$ (normalized to initial value) along the channel for $P_0^* = 0.4$ MPa. As it can be seen from Fig. 7, the main factors are the mass removal $\overline{G}_m$ and the friction $\overline{G}_f$ (and they have different signs $\overline{G}_m < 0$ and $\overline{G}_f > 0$) along the whole channel length. At the initial section $\overline{G}_f > |\overline{G}_m|$ and $\overline{G} > 0$. Further, downstream the amount of sucked air decreases (see Fig. 2) and $\overline{G}_f$ also decreases (in absolute value). At the section $x/d_h \approx 27$ the value of the function is $\overline{G} = 0$ and Mach number takes the critical value $\mathrm{M} = 1$ according to Eq. (12).

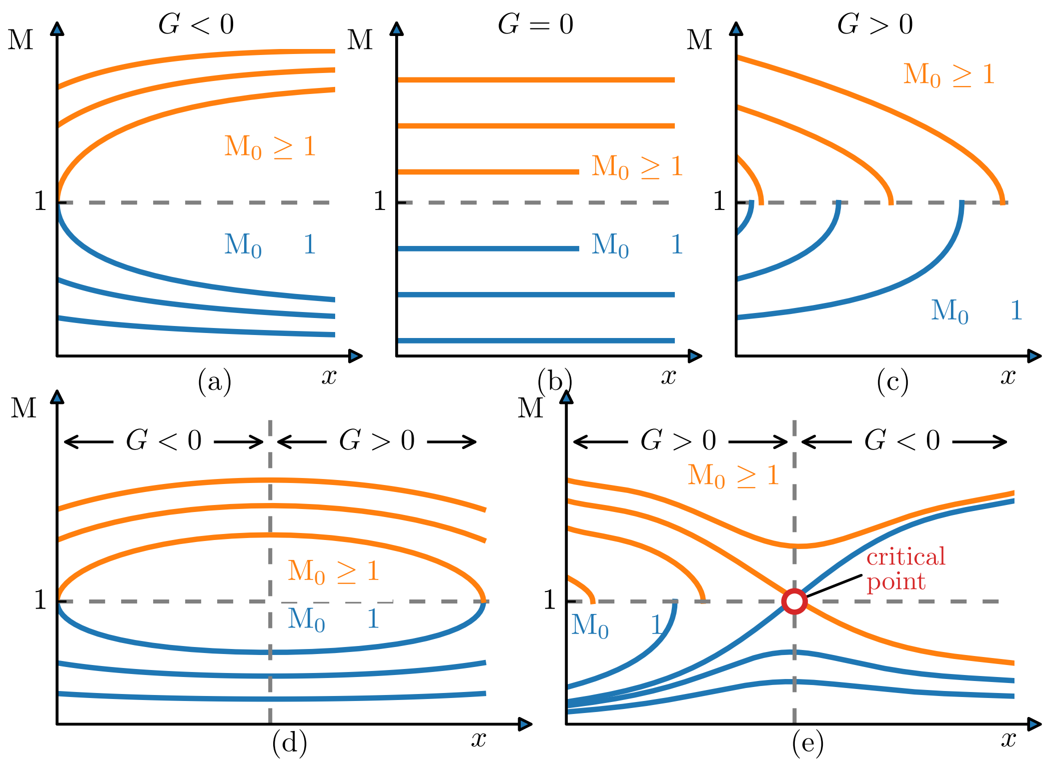


Figure 6. Possible options for changing the Mach number along a channel depending on the initial Mach number M_0 and the summary action G [10]

Summary action $\overline{G}$ changes its sign from positive to negative when passing through the critical section ($M = 1$).

Thus, the performed analysis shows that the shock-free deceleration in a constant cross-section channel with friction and suction through the permeable wall is possible. Moreover, the measured static pressure distribution (Fig. 2) and Mach number value at $x/d_h = 41.4$ (Fig. 4) confirm this conclusion.

It is possible to change the critical point location by using a tube with the different inner surface roughness. Figure 8 shows the local Mach number distribution (2D results) for different values of relative roughness. As it can be seen from the figure, a critical point locates upstream (closer to the initial section) for the bigger values of roughness. Then, as the roughness value decreases, the critical point shifts downstream. It can be noted that for the hydrodynamically smooth wall ($\Delta_s = 0$), there is no critical point and the pure supersonic flow is realized along the full channel length.

Conclusion

The flow in a supersonic nozzle with a permeable tube is considered. Two numerical models (one- and two-dimensional) have been developed that describe the processes occurring in such a device. The developed numerical models were validated against available experimental data. The experimental and calculated values are compared both along the channel axis and depending on the radius.

The presented results of numerical simulation of the flow in a permeable tube make it possible to conclude that a shock-free deceleration from supersonic to subsonic flow in a channel

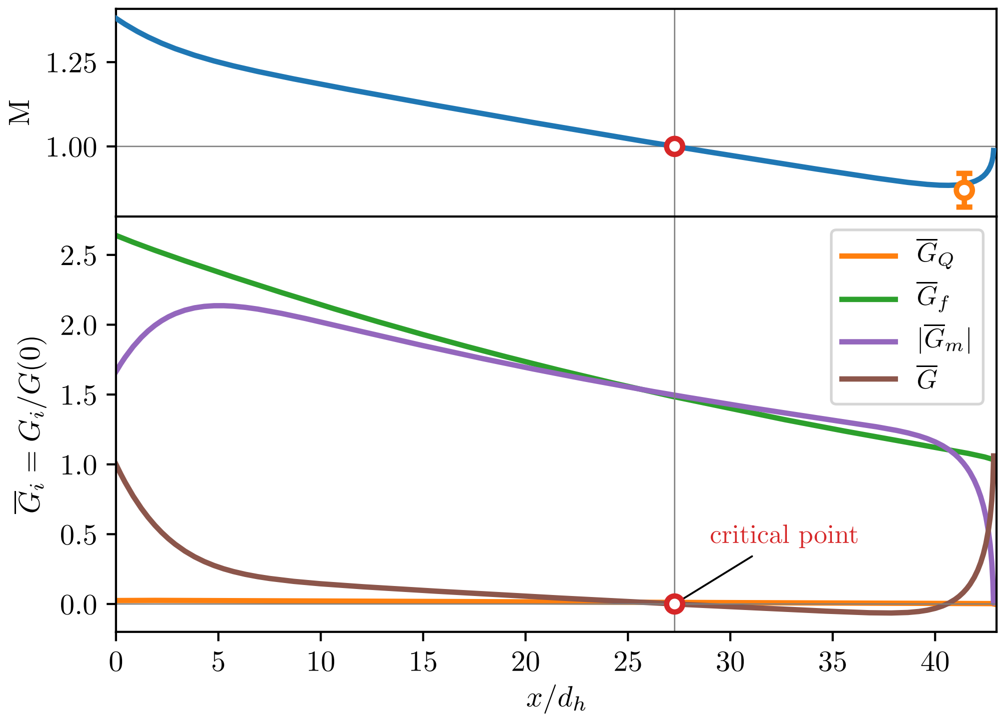


Figure 7. Changing of the Mach number and components of the function $\overline{G}$ along the channel with permeable wall. $P_0^* = 0.4$ MPa. Note, that for the presented case $\overline{G}_m$ is negative

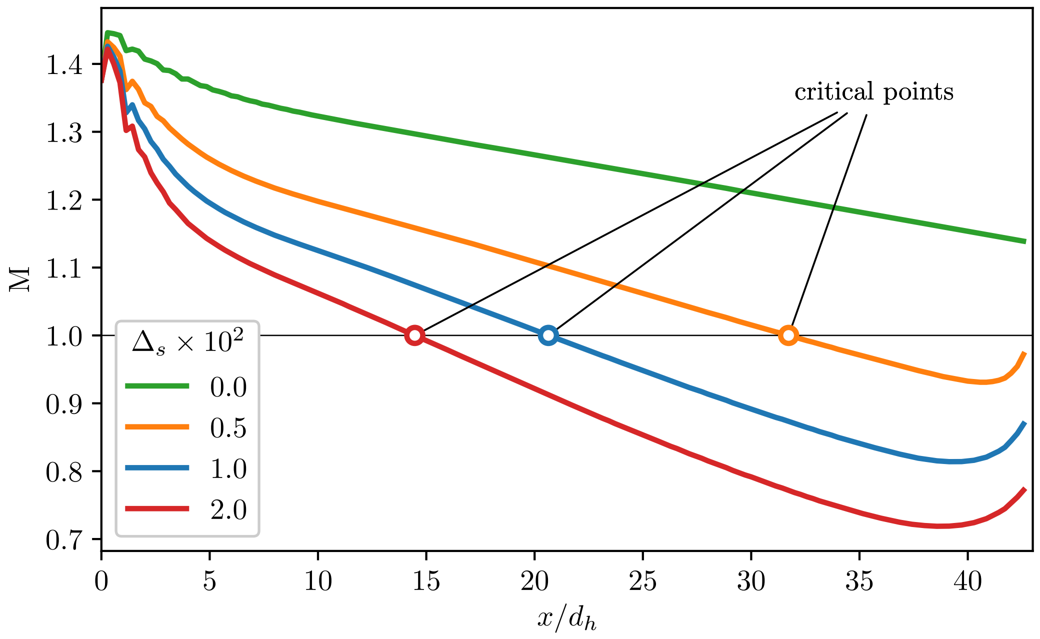


Figure 8. Influence of relative roughness on critical point location. $P_0^* = 0.4$ MPa

with permeable walls is possible. It is shown that the position of the transition (critical) point is determined by the pressure in the plenum and the relative roughness value. The comparison of the results of one-dimensional and two-dimensional models allows to conclude that the one-dimensional model is applicable to the analysis of such class of flows.

Acknowledgements

The work was performed according to the State Order of Lomonosov Moscow State University State Registration No. AAAAA19-119012990115-5.

The author expresses his gratitude to Ph.D. A.G. Zditovets for valuable comments during the discussion, regarding article materials.

References

1. Emmons, H.: Fundamentals of gas dynamics. High Speed Aerodynamics and Jet Propulsion, Princeton University Press (1958)

2. Geuzaine, C., Remacle, J.F.: Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering 79(11), 1309–1331 (Sep 2009). `https://doi.org/10.1002/nme.2579`

3. Grodzovskii, G., Nikol'skii, A., Svishchev, G., Taganov, G.: Supersonic gas flows in perforated boundaries. Mashinostroenie (1967)

4. Gus'kov, O.V., Kopchenov, V.I., Lipatov, I.I., *et al.*: Stagnation processes of supersonic flows in channels. Fizmatlit (2008)

5. Khazov, D.E.: On the question of gas-dynamic temperature stratification device optimization. Journal of Physics: Conference Series 891(1), 012078 (2017). `https://doi.org/10.1088/1742-6596/891/1/012078`

6. Kutateladze, S.S., Leontiev, A.I.: Turbulent Boundary Layers in Compressible Gases. Academic Press and Arnold (1964), (translated and exquisitely commented by D.B. Spalding)

7. Leontiev, A.I., Volchkov, E.P., Lebedev, V.P.: Thermal protection of plasmatron walls. Low temperature plasma, vol. 15. In-t teplofiziki SO RAN, Novosibirsk (1995)

8. Leontiev, A.I., Zditovets, A.G., Kiselev, N.A., *et al.*: Experimental investigation of energy (temperature) separation of a high-velocity air flow in a cylindrical channel with a permeable wall. Experimental Thermal and Fluid Science 105, 206–215 (2019). `https://doi.org/10.1016/j.expthermflusci.2019.04.002`

9. Rennels, D., Hudson, H.: Pipe Flow: A Practical and Comprehensive Guide. Wiley (2012)

10. Shapiro, A., H.: The dynamics and thermodynamics of compressible fluid flow, vol. 1. The Ronald Press Company (1953)

11. Vinogradov, Y., Leontev, A.: Gas flow in a supersonic axisymmetric nozzle with a permeable insert. Fluid Dynamics (5), 205–208 (1999)

12. Zditovets, A.G., Leontiev, A.I., Kiselev, N.A., *et al.*: Experimental study of the temperature separation of the air flow in a cylindrical channel with permeable walls. In: Proceedings of the 16th International Heat Transfer Conference, IHTC-16, Beijing, China (2018). `https://doi.org/10.1615/IHTC16.her.021878`

DOI: 10.14529/jsfi220403

Functional Programming Libraries for Graphics Accelerators

Mikhail M. Krasnov[1], ***Olga B. Feodoritova***[1]

Modern graphics accelerators (GPUs) can significantly speed up the execution of numerical tasks. However, porting programs to graphics accelerators is not an easy task, sometimes requiring their almost complete rewriting. CUDA graphics accelerators, thanks to the technology developed by NVIDIA, allow you to have a single source code for both conventional processors (CPUs) and graphics accelerators (CUDA). However, parallelization on shared memory is done differently and still must be specified explicitly. The use of the functional programming library developed by the authors makes it possible to hide the use of one or another parallelization mechanism on shared memory inside the library and make the user source code independent of the computing device used (CPU or CUDA). Functional programming is based on the modern mathematical theory, namely the Category Theory, in which the notions of Functors and Monads are widely used. Our work intensively utilizes these notions and extends them to grid expressions used in solving numerical problems.

Keywords: C++, functional programming library, CUDA, OpenMP, OpenCL, OpenACC.

Introduction

In recent years, graphics accelerators (GPUs) have become increasingly popular as computing devices for numerical calculations. Such accelerators are installed on many computing clusters. In the TOP500 list of the most productive supercomputers of November 2022 [1], graphics accelerators from NVIDIA [2] and AMD [3] occupy leading places, including the first place, which has overcome the long-desired exascale performance. NVIDIA, which for many years was among the first, has lost the palm to AMD with its AMD Instinct$^{\text{TM}}$ MI250X Accelerator. As for the most high-performance supercomputers in Russia, as of March 2022 [4], almost all of them are equipped with accelerators from NVIDIA. The speed of numerical calculations on such accelerators can be many times higher than on the CPU (according to the experience of the authors, the acceleration can reach 10–20 times), so the transfer of programs that implement numerical methods to graphics accelerators is an extremely urgent task.

However, porting an existing program to a GPU is not an easy task. Of course, the ideal option is to immediately write a program so that it can work on any computer. In any case, the main question arises – what technology to use for the GPU? Currently, there are three main technologies – OpenCL (an open standard for heterogeneous systems) [5], OpenACC [6] and CUDA, developed by NVIDIA for its graphics accelerators [7]. Each of these technologies has its own advantages and disadvantages. The main advantage of OpenCL is that it is an open standard. A program that uses OpenCL will run on any computing device that supports this standard, including NVIDIA and AMD GPUs, Intel Xeon Phi processors with Intel MIC technology, and even conventional CPUs. The main disadvantage of this technology is that the source code of the program often appears in two copies: for the CPU, which is compiled by a conventional compiler and is part of the main program, and the text for OpenCL in separate files. With changes in the algorithms, changes will need to be made in both places. The advantages and disadvantages of the CUDA technology are a mirror image of the advantages and disadvantages of OpenCL. CUDA works only on NVIDIA GPUs. On the other hand, in CUDA we have a single source code that is precompiled and is part of the main program (including the code that will be

[1]Keldysh Institute of Applied Mathematics, Moscow, Russian Federation

executed on the GPU). Unfortunately the OpenACC technology is not accessible to us, as the compiler that supports this technology is not installed on our clusters with graphics accelerators.

Nowadays, many calculations are carried out on heterogeneous systems using graphics accelerators. Examples include recent publications [9, 12, 13]. Some of works use OpenCL technology and as a result have two versions of source code, others use NVIDIA CUDA and have common code for CPU and GPU, but they cannot run on e.g. AMD GPUs. We choose CUDA technology. Our main argument is that in (our) real life we only deal with devices from NVIDIA. AMD GPUs and Intel Xeon Phi processors are still quite exotic for us. Therefore, the disadvantage of CUDA is not a disadvantage for us, but its advantage remains.

The next problem is that shared memory parallelization is done quite differently on the CPU and on the GPU. If we want to get a single text that should be compiled for both the CPU and CUDA, then in those places where there should be parallelization, we will have to write different code (for example, using the **#ifdef** construct), which is inconvenient. We emphasize once again that we are talking about the parallelization of the loops on shared memory. Although it is possible to write a single source code for the CPU and GPU for the loop body, the loop organization itself is written differently.

And then the idea arose to use the funcprog functional programming library for the C++ language [14], previously written by one of the authors of the article. An appropriate modification of this library will allow all the specifics of a computing device (CPU or CUDA) to be placed inside the library, and the user source code will turn out to be completely platform independent.

The article is organized as follows. Section 1 is devoted to a brief introduction to functional programming (to the extent necessary to understand the rest of the text). In Section 2 a brief description of the funcprog2 functional programming library is given and Section 3 contains examples of using this library to solve numerical problems. Conclusion summarizes the study and points directions for further work.

1. Functional Programming Library

1.1. Scope of the Funcprog2 Library

Let us describe a family of algorithms for which the developed funcprog2 library is applicable.

Technologically, any task of numerical simulation begins with the construction of a grid in the calculation area. Moreover, in areas of complex shape, this grid is, as a rule, non-structural. The grid functions of interest to the researcher can be specified both at the nodes of the cells and at their centers.

At this stage of the study, we consider only non-stationary problems and believe that various explicit schemes are used for time integration, for example, in the software package that we used to transfer to the GPU, this is an explicit classical Runge–Kutta scheme of the fourth order. Implicit schemes involving the solution of linear systems of equations have not been considered at the moment. Using the described approach to solve settling problems using implicit schemes requires additional developments that expand the capabilities of the presented library.

If the method is explicit, then the values of the grid functions at different points of the grid can be calculated independently of each other, and, therefore, these calculations can be carried out in parallel. Thus, each explicit step (loop over the grid function index) can be parallelized on shared memory. Note that MPI parallelization is also possible, but has not yet been considered.

The developed functional programming library funcprog2 allows an applied mathematician to implement the described numerical algorithms without delving into the details and features of parallelization.

1.2. General Description of the Funcprog2 Library

When implementing the functional programming library funcprog2 for the C++ language, the task was to write a library with which one could write in the C++ language in a style close to the style of the Haskell language [8]. We cite the Haskell language as a role model, as it seems to be one of the most advanced modern functional programming languages based on modern mathematical theory (category theory), widely used and actively supported by the world scientific community. Details about category theory can be found, for example, in the books [15, 16].

An important question is what is a function from the point of view of this library? In the original version of the library, a function meant an object of the std::function class. This option does not suit us now, since we want the function to be executed on the graphics accelerator, and the std::function class object can only be executed on the CPU (mainly because its implementation uses virtual functions that are not portable on the GPU). It cannot be an ordinary function either, since it cannot be passed as a parameter from the CPU to CUDA, because an ordinary function can only be passed by address, and addresses cannot be passed from the CPU to CUDA. It was decided to create a function2 class within the funcprog2 library and consider any object of this class to be a function. Any function object (having a functional operator ()) can be converted into an object of the function2 class, in particular, it can be a lambda expression. Recall that in modern C++ (since C++11), a lambda expression begins with a pair of square brackets, inside of which variables accessible from the lambda expression can be placed. In order for an object to be passed to CUDA, the functional operator must be marked with the __device__ keyword. To make user source code platform-independent, the funcprog2 library has a __DEVICE macro, which expands to the __device__ keyword when compiled for CUDA, and to an empty string when compiled for CPU. Thus, a functional operator must be marked with the word __DEVICE. To convert a functional object into an object of the function2 class, the library has a special function _ (underscore). Here is an example of working with the funcprog2 library:

```
double d=(_([](double x){ return x * x; }) &
  _([](double x){ return x + 1; }))(5); //36
```

In this example, we created two functions, composed them (using the & operator), and invoked the resulting compound function with a parameter of 5. The result is 36.

1.3. Implementation of Functors, Applicatives and Monads

The funcprog2 library essentially relies on the concepts of functor, applicative, and monad. The implementation of functors, applicatives, and monads in the library is similar to the implementation of these concepts in Haskell language. Any class can declare itself a functor, an applicative, or a monad. To do this, it is enough to implement a specialization of the Functor, Applicative and Monad classes, respectively, for this class. You don't need to make any changes to the class itself.

Inside the specialization of the Functor class, you need to define a static function fmap. The specializations of the Applicative and Monad classes are defined similarly. For an applicative, the methods are called pure and apply, and for a monad, mreturn and bind. When implementing the static methods of these classes, one should not forget about the implementation of functor, applicative and monadic laws. Note also that in the funcprog2 library, the division operator is used as a functor operator, and multiplication is used as an applicative operator.

1.4. Grid Expressions and Grid Functions

The notion of a grid expression plays a significant role in the funcprog2 library. This is an object defined for all grid indices, that is, for any object that is a grid expression, you can find out what its value is for a given index. The simplest special case of a grid expression is a grid function, which simply stores its values in memory and returns them, if necessary. For grid expressions, a grid_expression class template is defined, from which all classes of objects that are grid expressions must inherit (in particular, the grid_function class is also inherited from the grid_expression class). Thus, the phrase “an object is a grid expression” means that the class of this object is inherited from the grid_expression class. This inheritance uses expression templates [18] and the CRTP (Curiously Recurring Template Pattern) [10] design pattern, in which the final class is passed to the base class as a template parameter.

Any grid expression can be assigned to a grid function. This assignment operator iterates over all indices of the grid function to which the grid expression is assigned, for each index queries the grid expression for its value, and assigns that value to the grid function at the given index. The assignment operator implies that values for different indices can be calculated independently of each other, and therefore they can be calculated in parallel. It is in the assignment operator that the inner loop over the elements of the grid function is performed. The method of parallelization of this loop is chosen by the assignment operator, depending on which compiler the program is compiled with. If this is a compiler for CUDA (the __CUDACC__ preprocessor variable is defined), then parallelization is performed using CUDA, otherwise, using OpenMP. Thus, the parallelization method is hidden from the application programmer within this assignment statement.

Speaking about grid functions, one more aspect should be mentioned. The GPU can only work with its own memory, which means that when working on the GPU, the grid function must request memory for its data in CUDA memory. There are no problems with this either. Grid functions are arranged in such a way that when compiled on CUDA they request memory from CUDA, otherwise they request memory from the CPU.

1.5. Grid Expressions as Functors, Applicatives and Monads

Grid expressions can be thought of as containers (this is especially true for grid functions). In the funcprog2 library, containers (such as lists) are functors, applicatives, and monads. This makes it possible to apply ordinary functions to the values stored in them (a property of functors). Let’s make the grid expression also a functor, an applicative, and a monad so that functions can be applied to grid expressions as well. To understand how this can be done, consider a typical loop that calculates the new value of the grid function from the old one:

```
for(size_t i = 0; i < N; ++i)
  f_new[i] = calculate(f_old[i]);
```

Here *calculate* is a function that calculates the new value in the cell according to the old one. It is passed the old value in the cell as a parameter. In the new approach, we want to be able to write something like this in this case:

```
f_new = _(calculate) / f_old;
```

If the calculations require several more grid functions (let's call them f2 and f3), then instead of

```
for(size_t i = 0; i < N; ++i)
  f_new[i] = calculate(f_old[i], f2[i], f3[i]);
```

we could write:

```
f_new = _(calculate) / f_old * f2 * f3;
```

that is, for the first grid function, we applied the functor property, and for the subsequent ones, we applied the applicative. If we want to pass some additional constant value to the function (independent of the cycle index), then instead of

```
for(size_t i = 0; i < N; ++i)
  f_new[i] = calculate(f_old[i], some_value);
```

we could write

```
f_new = _(calculate) / f_old * pure(some_value);
```

Thus, the result of applying a function to a grid expression (or to several grid expressions in the case of an applicative) must also be a grid expression, that is, it can be asked for a value by index (the [] operator must be implemented). Grid expressions, in addition to grid functions, are also the results of applying functions to grid expressions as functors and monads. In addition, the sum and difference of two grid expressions, as well as the product and quotient of a grid expression and a number, are also grid expressions.

Let's show how functors, applicatives and monads for grid expressions are implemented.

Functor. The *fmap* function takes a function with one parameter and a functor (a grid expression in our case) and returns the same functor (a new grid expression). This new grid expression stores the parameters of the fmap function in its class member variables (let's call them *f* and *gexp*) and implements the [] operator as follows (in pseudo-Haskell):

```
(fmap f gexp)[i] = f gexp[i]
```

Applicative. The *pure* function takes some value and "introduces" it into the applicative. In our case, it makes a grid expression out of it. Let's define its operator [] so that it returns the same value val for any index:

```
(pure val)[i] = val
```

The *apply* function (an analogue of the (<∗>) operator in Haskell) in our case takes two grid expressions: the first (let's call it gexp_f) returns functions, and the second (let's call it gexp) returns some values (the parameters of these functions). Let's define the grid expression of the *apply* function as follows:

```
(apply gexp_f gexp)[i] = gexp_f[i] gexp[i]
```

Monad. The *return* monad function is defined in the same way as the *pure* applicative function:

```
return = pure
```

The *bind* monad operation (in the Haskell language and in the funcprog2 library, the (>>=) operator) takes a monad (in our case, a grid expression, let's denote it by the variable *gexp*) and a function that takes a regular (non-monadic) value and returns a monad (in our case, a grid expression). Let's define the *bind* operation as follows:

```
(bind gexp f)[i] = (f gexp[i])[i]
```

We have proved three theorems that these definitions of the functor, applicative, and monad for grid expressions are correct, that is, they satisfy the functor, applicative, and monad laws, respectively. This article does not present this evidence.

So grid expressions are functors, applicatives, and monads. This means that any ordinary unary function can be "applied" to a grid expression (using the *fmap* function). This application will return a new grid expression. Any binary function can be "applied" to two grid expressions (using the *apply* function), and any function with n arguments can be "applied" to n grid expressions. This can be done in one line. For example, suppose there are two grid functions f and g. Then you can write like this:

```
g = _([](double x){ return sin(x); }) / f;
```

Since grid expressions are monads, it is possible to build chains of monad calculations of the form from them:

```
g = f >>= f1 >>= f2 >>= f3;
```

Here f1, f2, f3 are some functions that take ordinary (non-monadic) values and return grid expressions.

2. Examples

2.1. The Simplest Example

Consider the *axpy* function from the BLAS library. This function takes a constant a and two vectors (x and y) and modifies the vector y as y[i] += a ∗ x[i]. Its implementation for a conventional processor can be written as follows:

```
template<typename T>
void axpy(T a, vector<T> const& x, vector &y){
#pragma omp parallel for
  for(int i = 0; i < y.size(); ++i)
    y[i] += a*x[i];
}
```

This function is perfectly parallelized, but the method of parallelization in this implementation is specified explicitly and is not suitable for graphics accelerators. Using functional programming, this function could be rewritten as follows:

```
template<typename T>
void axpy(T a, grid_function<T> const& x, grid_function<T> &y){
  y = _([](T a, T xi, T &yi, size_t /*i*/){
    yi += a * xi;
  }) / pure(a) * x;
}
```

The lambda expression in the body of this function takes as parameters our constant a, the i-th element of the grid function x, by reference the i-th element of the grid function y, and the current loop index i (which we ignore). Each input parameter (and we have two of them) corresponds to one parameter of the lambda expression, and the grid function, to which the expression is assigned, corresponds to the output parameter (passed by reference) and the loop index. The first parameter (in our case it is pure(a)) is passed as for the functor (via the / operator), and the next as for the applicative (via the * operator). Here is an example of calling the *axpy* function:

```
int main(){
  size_t const N = 10;
  math_vector<double> x(N, 2), y(N, 3);
  axpy(5., x, y);
  std::cout << y[0] << std::endl; // 13
}
```

2.2. More Complex Example

Now we will calculate an expression like z[i]=a*x[i]+b*y[ind[i]] (let's call the function *axpby*). Its peculiarity is that the index of the grid function y is contained in the additional grid function *ind*:

```
template<typename T>
void axpby(T a, grid_function<T> const& x,
T b, grid_function<T> const& y,
grid_function<size_t> const& ind, grid_function<T> &z)
{
  z = _([](T a, T xi, T b, grid_function_proxy<T> const y_p,
    size_t indi, T &zi, size_t /*i*/)
  {
    zi = a * xi + b * y_p[indi];
  }) / pure(a) * x * pure(b) * pr(y) * ind;
}
```

What is essentially new compared to the first example is that we can no longer pass the grid function y through *apply*. Instead, we need to create a proxy object for it and pass it through pure. This is exactly what the *pr* library function does:

```
template<class F>
auto pr(F const& f){
  return pure(get_proxy(f));
}
```

A few words about proxy objects. They are needed to transfer entire grid functions to the GPU. The fact is that in the GPU, parameters can only be passed by value, that is, their copy will be made. But a copy of the grid functions cannot be made, as this will lead to the creation of a copy of the data. Instead, a “light” placeholder object is created for the grid function, storing a pointer to the data and the size of the grid function. It is this object that is passed by value as a whole (using the pure function).

Let’s also pay attention to the fact that global variables that are stored in CPU memory are not available from the GPU, so they have to be passed explicitly using the pure function. If you need a buffer for intermediate calculations, then its size should be equal to the number of grid nodes, since in the case of CUDA we do not know the absolute number of the execution thread.

Another important problem that often arises is reduction (for example, finding the maximum value or the sum of all values). It is also very important to carry out the reduction in parallel. In the original version of the program, the reduction was performed by means of OpenMP, but now we cannot use this mechanism. Fortunately, in modern C++ (since the C++17 language standard), reduction functions (such as std::accumulate and std::reduce) have parallel versions. When working on CUDA, we use the thrust library, which is part of the CUDA Computing Toolkit. This library also has parallel reduction functions running on GPU. Thus, when working on CUDA, we use the parallel reduction functions from the thrust library, when working on the CPU, if there are parallel versions of the reduction functions (if the compiler supports them), then they are used, otherwise the reduction is done sequentially.

3. Performance Comparison

To evaluate the effectiveness of the proposed approach, a problem is proposed that simulates an experiment carried out in the L2K pipe (Germany, [17]). An axisymmetric body consisting of two conical and cylindrical surfaces is placed in an oncoming air flow with the following characteristics: $M_\infty = 4.7$, $p_\infty = 272Pa$, $T_\infty = 764K$, $Y_{O2} = 0.245$, $Y_{N2} = 0.755$. The geometry of the problem is shown in Fig. 1. The solid body is combined from two parts with different thermodynamic characteristics: the head part is made of UHTC (Ultra High Temperature Material) and the rear cylindrical part is copper.

The initial temperature of the body is T=300 K. The adjoint problem is solved in a two-dimensional formulation. Modeling is based on the solution of the Navier–Stokes equations in a multicomponent gas and the heat equation in a solid using an original technique based on explicit Chebyshev iterations [11]. The mesh size is 297192 nodes.

We solved this problem using a K60gpu hybrid supercomputer installed at the Keldysh Institute of Applied Mathematics of RAS. One node of this supercomputer has two host processors Intel Xeon Gold 6142 v4, 16 cores (total 32 threads) and four GPUs nVidia Volta GV100GL. We used the program complex NOISETTE [12], which originally can work in two modes of parallelization: on CPU using OpenMP and on using OpenCL (with separate sources for kernels). We have rewritten it using our approach. For testing on CPU mode we used one host (32 threads) and for OpenCL and CUDA modes – one GPU. The acceleration compared to the CPU when using OpenCL was about 20 times, and when using our approach – about 12 times. The acceleration is not as great as when using OpenCL, but given the above advantages (above all, a single source code), it can be considered acceptable.

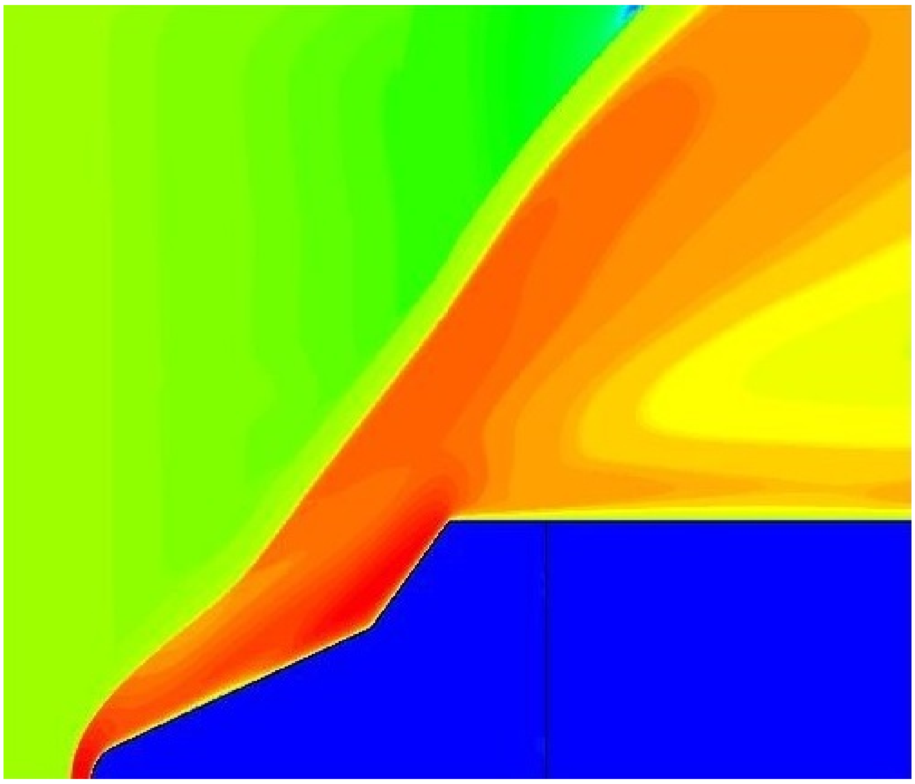


Figure 1. Geometry of the problem with the shock wave structure

Conclusion

Declarative programming languages, which include functional languages, allow, unlike imperative languages, which include most programming languages in which numerical methods are implemented, to briefly and at the same time quite clearly write down the desired result without going into implementation details. The specific implementation may be hidden in the language and depend on the current hardware and software environment. The C++ language proved to be powerful enough to allow the implementation of a functional programming library on it, which allows you to write programs in a style close to the style of purely functional languages such as Haskell. Such concepts from the world of functional programming as functors and monads, implemented in the functional programming library, turned out to be a very convenient tool for porting numerical problems to CUDA graphics accelerators. Grid expressions have been defined as functors, applicatives, and monads, allowing functions to be applied to the values they store. These functions themselves can be built by combining complex functions from simple ones, which is also the strength of functional programming.

References

1. TOP 500. `https://www.top500.org`

2. NVIDIA. `https://www.nvidia.com`

3. AMD. https://www.amd.com
4. TOP 50. http://top50.supercomputers.ru
5. OpenCL. https://www.khronos.org/opencl
6. OpenACC. https://www.openacc.org
7. CUDA Zone. https://developer.nvidia.com/cuda-zone
8. Haskell language. https://www.haskell.org
9. Chaplygin, A., Gusev, A., Diansky, N.: High-performance shallow water model for use on massively parallel and heterogeneous computing systems. Supercomputing Frontiers and Innovations 8(4), 74–93 (2021). https://doi.org/10.14529/jsfi210407
10. Coplien, J.O.: Curiously recurring template patterns. C++ Report pp. 24–27 (February 1995)
11. Feodoritova, O., Krasnov, M., Zhukov, V.: A numerical method for conjugate heat transfer problems in multicomponent flows. J. Phys.: Conf. Ser 2028 012024 (2021). https://doi.org/10.1088/1742-6596/2028/1/012024
12. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271(108231) (February 2022). https://doi.org/10.1016/j.cpc.2021.108231
13. Gorobets, A., Duben, A.: Technology for supercomputer simulation of turbulent flowsin the good new days of exascale computing. Supercomputing Frontiers and Innovations 8(4), 4–10 (2021). https://doi.org/10.14529/jsfi210401
14. Krasnov, M.M.: Functional programming library for C++. Programming and Computer Software 46, 330–340 (2020). https://doi.org/10.1134/S0361768820050047
15. MacLane, S.: Categories for the Working Matematitian. Springer (1998)
16. Milewski, B.: Category Theory for Programmers (2019), https://github.com/hmemcpy/milewski-ctfp-pdf/releases/download/v1.3.0/category-theory-for-programmers.pdf
17. Murty, C., Manna, P., Chakraborty, D.: Conjugate heat transfer analysis in high speed flows. Proceedings of Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 227(10), 1672–1681 (2013). https://doi.org/10.1177/0954410012464920
18. Veldhuizen, T.: Expression templates. C++ Report 7(5), 26–31 (June 1995)

DOI: 10.14529/jsfi220404

Application of the Nonlinear SST Turbulence Model for Simulation of Anisotropic Flows

Andrey A. Savelyev[1], ***Innokentiy A. Kursakov***, ***Evgeniy S. Matyash***[1], ***Evgeny V. Streltsov***[1], ***Ruslan A. Shtin***[1]

The application of the nonlinear SST turbulence model (SST NL) for the calculation of flows with turbulence anisotropy is considered. The results of the following validation test cases are presented: the flow in a square duct, the corner flow separation at a wing-body junction (NASA Juncture Flow) and the transonic wing flow (NASA CRM). The nonlinear model has been found to significantly improve the quality of simulating the anisotropic flows as compared to models based on the Boussinesq hypothesis. It is shown that the model prevents false corner separation at the wing-body junction and thereby achieves a qualitative improvement in simulation results. The test case of the transonic wing flow revealed an upstream displacement of the shock wave on the upper side of the wing which leads to an underestimation of the lift force when using the SST NL model. In all the tests considered, the SST NL model required an increase in computational cost of at most 5 % compared to the conventional SST model.

Keywords: turbulence model, nonlinear SST, SST NL, turbulence anisotropy, corner flow, corner separation.

Introduction

The main tool currently used in studies of aircraft aerodynamics is the numerical solution of the Reynolds-averaged Navier–Stokes equations, closed by a turbulence model. The most common turbulence models are the Spalart–Allmaras (SA) [1, 21], Menter (SST) [11, 12] and their modifications (e.g., [8]). Both models belong to the family of linear turbulent viscosity models (LEVM) based on the Boussinesq hypothesis, which assumes a linear relationship between the Reynolds stress tensor and the mean velocity gradient. This limits their applicability to the situations in which the flow is mainly affected by a single component of the stress tensor. For more complex flows (e.g., corner flows or separation flows), the effects of turbulence anisotropy appear that cannot be described by the Boussinesq hypothesis.

A practically important example of such a flow is the corner flow near the wing-fuselage junction: calculations using Boussinesq models predict extensive corner flow separation, even at small angles of attack. At higher values of lift on transonic regimes the separation starts interacting with the shock wave, which can lead to global changes in the pressure distribution on the upper surface of the wing and, consequently, to a noticeable reduction of the lift force. Such an effect, observed in the computational studies of the models DLR-F6 [14] and CRM [23], is not confirmed by experimental tests.

In order to correctly describe corner flows, it is necessary to use more complex turbulence models that can reproduce the anisotropy of the Reynolds stress tensor. These are primarily differential models for Reynolds stresses (DRSM) [7]. DRSM models require solving a specific transfer equation for each component of the Reynolds stress tensor. This noticeably increases the computational resource requirements, and makes it more difficult to formulate boundary conditions. In addition, the nonlinearity of these models can lead to a lack of numerical stability, especially when dealing with complex flows [4]. For these reasons, the DRSM models are rarely used in practical applications.

[1]Central Aerohydrodynamic Institute (TsAGI), Zhukovsky, Russia

Another approach is the EARSM [24] models, which use explicit algebraic relations between the Reynolds stresses and the strain rate tensor. In terms of physical processes simulating, the EARSM models are intermediate between the DRSM and LEVM approaches and can adequately describe some of the physical phenomena beyond the Boussinesq hypothesis.

Non-linear extensions of Boussinesq models (Non-linear Eddy Viscosity Models, NLEVM) can also be referred to as the Reynolds stress models, since their final stress tensor relations have the same form as in EARSM. The most popular in this family is a modification of the Spalart–Allmaras model – SA QCR [20], which uses a quadratic function from the strain rate and vorticity tensors to model the Reynolds stress tensor. This model, although slightly more complicated, allows a better simulation of corner flows. The present work considers a nonlinear modification of the SST turbulence model – the SST NL model proposed by Garbaruk and Matyushenko [9]. To determine the Reynolds stresses, it combines the linear part from the SST model and the nonlinear part from the S BSL EARSM model [10].

The purpose of this paper is to test the SST NL model both on model test cases and on configurations of interest for practical applications. The paper is organized as follows: Section 1 presents the formulation of the SST NL model [9] and a brief description of the EWT-TsAGI program used. Section 2 presents the results of the test calculations: turbulent flow in a square duct, wing-fuselage junction flow separation on the NASA Juncture Flow model [19], transonic wing flow on the NASA CRM model [13]. The conclusions are given in the final section.

1. Methodology

The expression for the Reynolds stresses in the SST model [11] has the following form:

$$\tau_{ij} = \frac{2}{3}k\delta_{ij} - 2\nu_t S_{ij}, \tag{1}$$

where $\nu_t = \dfrac{a_1 k}{\max\left(a_1\omega, SF_2\right)}$ is the turbulent eddy viscosity, $S = \sqrt{2S_{ij}S_{ij}}$ – strain rate invariant, F_2 – SST blending function, and $a_1 = 0.31$.

The S BSL EARSM model proposed in [10] is a development of the WJ EARSM [24]. The Reynolds stresses are defined using the anisotropy tensor:

$$\tau_{ij} = \frac{2}{3}k\delta_{ij} + ka_{ij}. \tag{2}$$

The anisotropy tensor is expanded in terms of the tensor basis as follows [15]:

$$a_{ij} = \beta_1 T_{1,ij} + \beta_3 T_{3,ij} + \beta_4 T_{4,ij} + \beta_6 T_{6,ij}. \tag{3}$$

According to [10], the tensor basis can be expressed in a modified (compared to [15]) form:

$$\begin{aligned} &T_{1,ij} = S^*_{ij},\ T_{3,ij} = \Omega^*_{ik}\Omega^*_{kj} - \tfrac{1}{3}II^*_\Omega\delta_{ij},\ T_{4,ij} = S^*_{ik}\Omega^*_{kj} - \Omega^*_{ik}S^*_{kj},\\ &T_{6,ij} = S^*_{ik}\Omega^*_{kl}\Omega^*_{lj} + \Omega^*_{ik}\Omega^*_{kl}T_{1,ij} - \tfrac{2}{3}IV^*\delta_{ij} - II^*_\Omega S^*_{ij}. \end{aligned} \tag{4}$$

Here $S^*_{ij} = \tau S_{ij}$ and $\Omega^*_{ij} = \tau\Omega_{ij}$ are dimensionless strain and vorticity tensors, and $\tau = \max\left[1/(\beta^*\omega), 6\sqrt{\nu/(\beta^* k\omega)}\right]$ is the turbulent time scale. The coefficients β are expressed through the tensor invariants $II^*_S = S^*_{mn}S^*_{nm}$, $II^*_\Omega = \Omega^*_{mn}\Omega^*_{nm}$, $IV^* = S^*_{mn}\Omega^*_{nk}\Omega^*_{km}$ as follows:

$$\beta_1 = -\frac{N}{Q}, \quad \beta_3 = -\frac{2IV^*}{NQ_1}, \quad \beta_4 = -\frac{1}{Q}, \quad \beta_6 = -\frac{N}{Q_1}, \tag{5}$$

where

$$Q = \left(N^2 - 2II_\Omega^*\right)/A_1, \quad Q_1 = Q\left(2N^2 - II_\Omega^*\right)/6, \quad N = C_1' + \tfrac{9}{4}\sqrt{2\beta^* II_S^*}. \tag{6}$$

The SST NL model [9] is a combination of the SST and S BSL EARSM models. Its expression for the Reynolds stresses τ_{ij} contains two terms: a linear term taken from the SST model (1), and a nonlinear term based on the S BSL EARSM model (2):

$$\tau_{ij} = \frac{2}{3}k\delta_{ij} - 2\nu_t S_{ij} + k\left(\beta_4 T_{4,ij} + \beta_6 T_{6,ij}\right). \tag{7}$$

Compared to the S BSL EARSM model, the nonlinear component has two simplifications: it does not use the term $T_{3,ij}$, and it uses the simplified expression $\tau = 1/(\beta^*\omega)$ as the turbulent time scale. The model constants appearing in (6) are $\beta^* = 0.09$, $A_1 = 1.8$, $C_1' = 1.8$.

The SST NL model is implemented in the TsAGI in-house CFD program Electronic Wind Tunnel (EWT-TsAGI [2]). In the presented simulations, a linearized implicit finite-volume scheme which is second-order in space and first-order in time is used for all equations. The convective fluxes are computed using the exact Godunov solution of the Riemann problem. Solution reconstruction in each cell is performed according to the MUSCL scheme. Diffusion fluxes are approximated by a modified second-order central difference scheme. The turbulence model source terms are computed using an unconditionally stable approximation analyzing the signs of the eigenvalues of the corresponding Jacobi matrix. More details on the numerical scheme, as well as validation results of EWT-TsAGI solver can be found in [3]. The code works with multi-block structured meshes and is parallelized using the MPI standard. All calculations discussed further in this paper have been performed using from 32 to 256 cores.

2. Test Cases

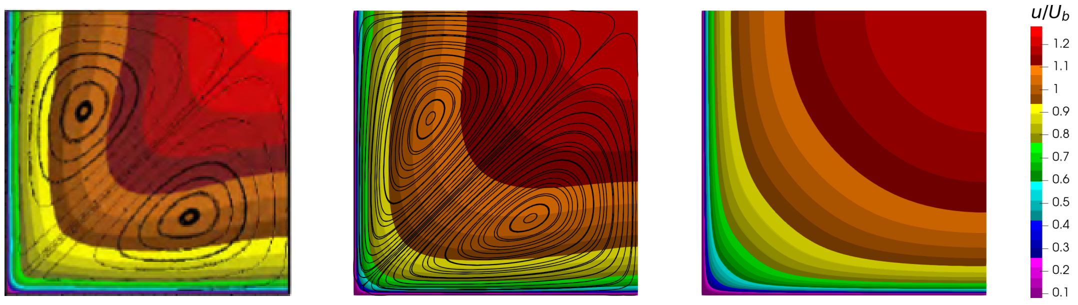
2.1. Flow in a Square Duct

A fully developed turbulent flow in a square duct is considered. The calculation of such a flow is a commonly used test for evaluating the ability of turbulence models to correctly describe the flow in corners, as it demonstrates the effect of turbulence anisotropy on the main flow, leading to significant changes in the flow structure. The width and height of the duct are 100 mm, the flow has the following parameters: density $\rho = 1.2\,\mathrm{kg/m^3}$, pressure $p = 101325\,\mathrm{Pa}$, temperature $T = 294.15\,\mathrm{K}$, average velocity $U = 3\,\mathrm{m/s}$. The Reynolds number, calculated from the average velocity and duct width, is $Re_b = 21400$. Simulation is performed for the duct length $L = 600\,\mathrm{mm}$ with periodic boundary conditions that combine the inlet and outlet of the duct. To maintain a steady flow and compensate for the gas momentum losses caused by wall friction, a uniform pressure gradient is added by introducing an additional term into the equation system being solved. To compensate for the heating caused by friction, a constant wall temperature is set equal to the initial gas temperature $T_w = T = 294.15$ K. A structured grid containing 240 cells across the duct uniformly spaced ($y^+ \approx 1.25$) and 480 cells along the duct were used for the simulations. Due to symmetric formulation of the task, a quarter of the duct was simulated.

The calculations performed using the linear SST model, its non-linear modification SST NL and the DRSM model are compared with the results of the authors of the SST NL model [5] and DNS data [5, 16]. The differential Reynolds stress model used for the simulations was SSG/LRR-ω [4]. This model combines the Speziale–Sarkar–Gatski pressure-strain model in free turbulent regions with the Launder–Reece–Rodi formulation near the wall by means of a blending function similar to that in SST model. The SSG/LRR-ω model was designed primarily for aerodynamic applications in the cases where anisotropic and three-dimensional effects are important.

The qualitative assessment of the simulation of secondary flows is performed by comparing the pattern of isolines of the longitudinal velocity profile in the duct cross section. Quantitative estimation is performed by comparing the longitudinal and transverse velocity profiles along the diagonal section of the duct. In Fig. 1 the numerical results of the considered flow using the SST and SST NL turbulence models are compared with the DNS results [5, 16]. The contours of the longitudinal velocity component and streamlines of a secondary flow are shown. It can be seen that the linear SST model significantly distorts the field of the longitudinal velocity component due to the absence of a secondary flow in the solution (see Fig. 1c). At the same time, the results obtained using the SST NL model predict the presence of a secondary flow and describe the longitudinal velocity field much better (Fig. 1b).

Figure 2 shows the distributions of the longitudinal and transverse velocity components along the diagonal of the duct cross section, which characterize the accuracy of the secondary flow intensity prediction. The figure compares EWT data (SST, SST NL, DRSM) with the results of the SST NL authors [5] and DNS data [16]. The graphs demonstrate that despite the relatively low values of the secondary flow velocity ($\sim 1\,\%$ of the average flow rate), it has a considerable influence on the flow structure. At the same time, the SST NL turbulence model provides a significant increase in the accuracy of the main flow calculation in comparison with the linear SST model. The difference of the longitudinal component of the flow velocity for the DNS and SST NL does not exceed $5\,\%$ and this difference reaches $20\,\%$ in the case of SST.



(a) DNS (b) SST NL (c) SST

Figure 1. Comparison of longitudinal velocity distributions obtained using SST and SST NL turbulence models with DNS results [5]

2.2. Separated Flow in the Wing-body Junction

Validation of the SST NL turbulence model implementation was carried out by computing a separated flow at the NASA Juncture Flow model. This test confirms the ability of the tested turbulence model to correctly predict the presence of separation at the wing-body junction of an aircraft, as well as its dimensions. The model geometry and the experimental results are available on the official website [19]. In NASA experimental work, a detailed investigation of the flow around the model with a shortened DLR F6 wing (fuselage length of 4.84 m, wingspan of 3.4 m, see Fig. 3a) was performed to verify numerical methods within the Juncture Flow workshop. These experiments investigated separation generation at the wing-fuselage junction

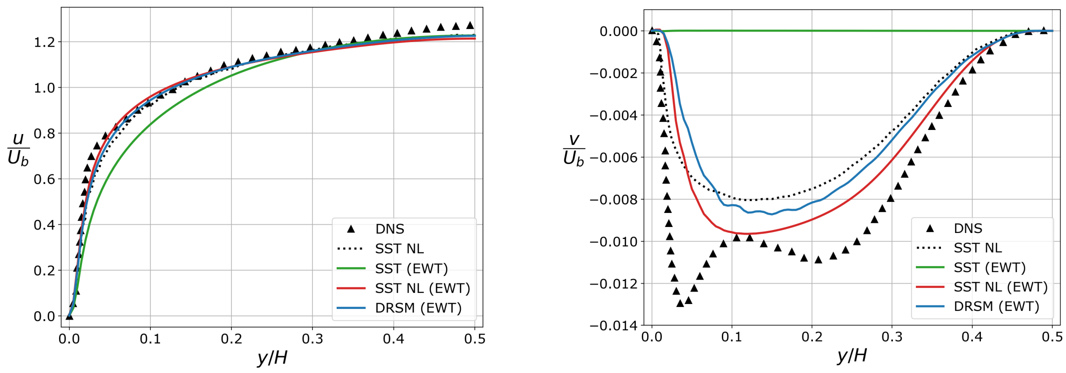
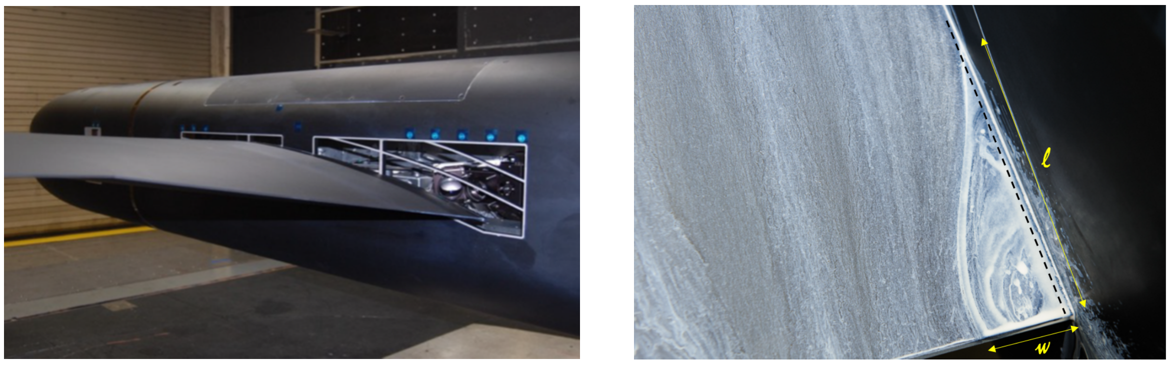


Figure 2. Distributions of the longitudinal and transverse velocity components along the diagonal of the duct cross section (DNS [16], SST NL [5], EWT)

and measured the longitudinal and transverse dimensions of the separation regions in various flow regimes, see Fig. 3b.

A multiblock structured grid with 85 million cells was used in calculations (Fig. 4). There were 324 cells per wing chord, boundary layer resolution was 64 cells, and the height of the first near-wall cell was $5 \cdot 10^{-6}$ m ($y^+ \approx 1.0$). Because of the symmetry of the problem, only half of the geometry was used. To estimate the errors associated with the discretization of the computational domain, the calculations were performed on a series of 3 grids generated from the base one by coarsening by a factor of 2 and 4 in each index direction. The following free flow parameters are considered: velocity $U = 64.4$ m/s, angle of the attack $\alpha = 5°$, pressure $p = 99000$ Pa, temperature $T = 288.84$ K. The quality of the corner flow simulation is assessed by comparing the numerical and experimental sizes of the separation region in the wing root.



(a) General view of the model (b) The separation at the wing root

Figure 3. NASA Juncture Flow test case [6]

Figure 5 shows the friction coefficient C_f distributions obtained with the linear and non-linear SST models. The linear version of the SST model predicts a developed diffuser separation at the wing-fuselage junction. In calculations with the SST NL turbulence model, the size of the separation region is much smaller compared to the SST.

Figure 6 shows the predicted separation sizes (length and width) as a function of $N^{-2/3}$, where N represents the number of grid cells. The power $-2/3$ refers to the second-order spatial accuracy: the results should vary linearly as the grid is refined ($N^{-2/3} \to 0$). The plots show that the solution predicted using the SST model depends on the grid resolution. The separation zone grows as the grid resolution increases, so on the finest grid, the separation size is significantly

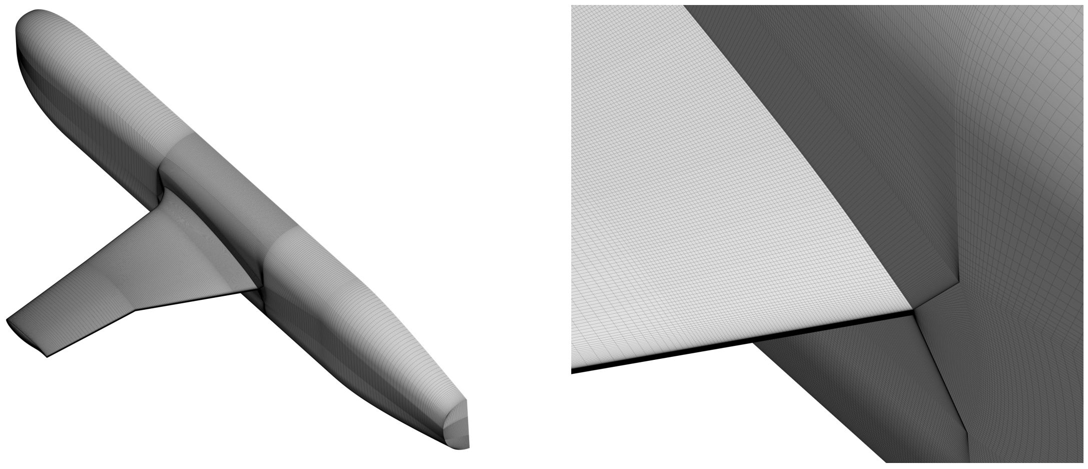


Figure 4. The surface mesh of the Juncture Flow model

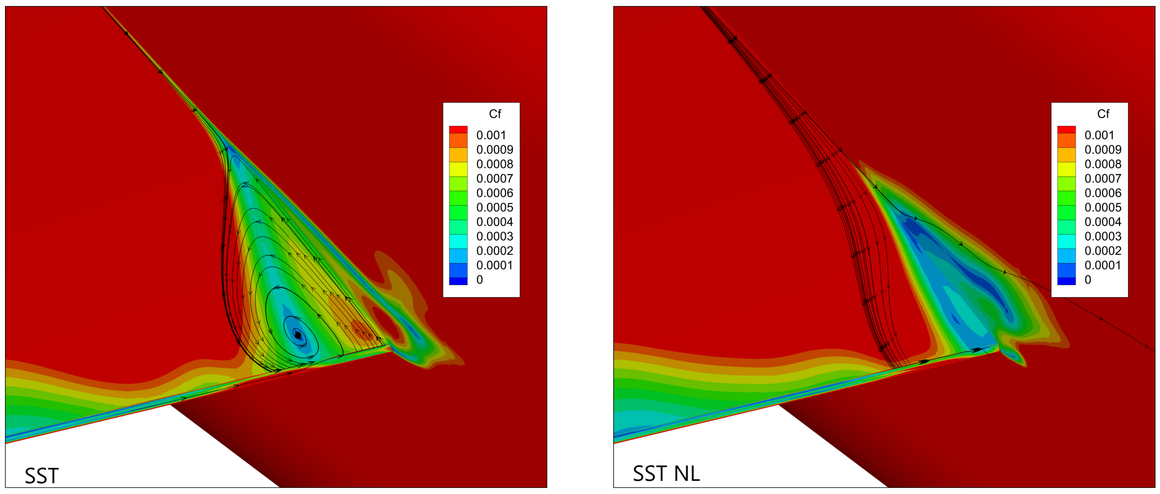


Figure 5. Distributions of the friction coefficient C_f obtained for the linear and nonlinear SST models

overestimated comparing to the experimental results [18]. The SST NL model, in general, permits to predict the longitudinal size of the separation zone more accurately: the discrepancy of numerical and experimental data is less than the experimental inaccuracy. However, the width of the separation zone in SST NL calculations with the finest grid is significantly underestimated. This fact requires additional computational investigations to study the influence of the grid resolution.

2.3. Transonic Wing Flow

Validation of the SST NL turbulence model at transonic flow regimes was performed on the well-known NASA Common Research Model (CRM). The model was developed for wind tunnel testing (NASA AMES 11-foot transonic wind tunnel, NASA National Transonic Facility). As a result, a database of experimental results has been obtained, that is used in CFD code validation benchmarks, such as AIAA Drag Prediction Workshop (DPW). The model geometry as well as the results of the experiments in different wind tunnels are available and can be found on the official website [17].

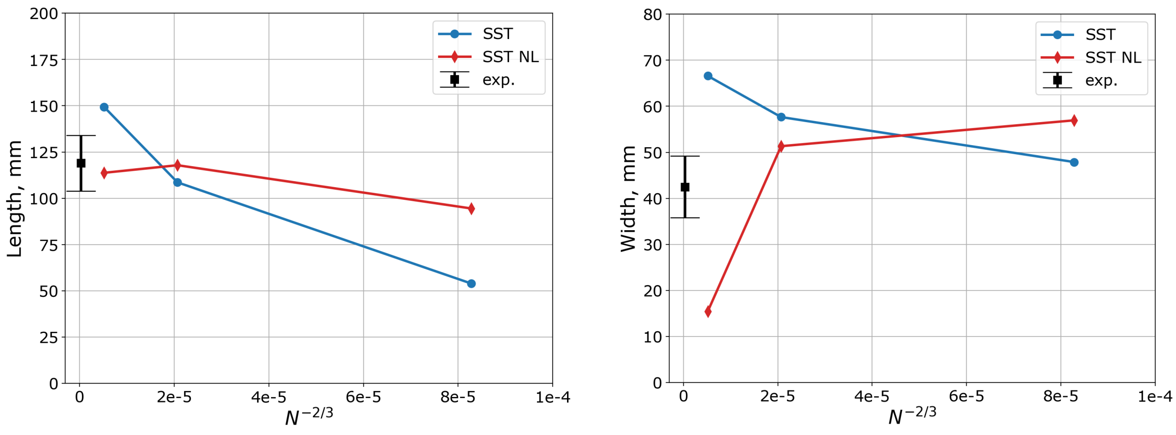


Figure 6. Predicted size of separation region at $\alpha = 5°$ on different grids (N is the number of grid cells) in comparison with experimental data [18]

The CRM model consists of a contemporary supercritical transonic wing and a fuselage that is representative of a widebody commercial transport aircraft. The model is designed for a cruise Mach number of $M_\infty = 0.85$ and a corresponding design lift coefficient $C_L = 0.5$. Leading edge sweep angle is 35°, the wing reference area is $S = 0.280$ m^2, the wing span is $b = 1.5866$ m, and the mean aerodynamic chord is $c = 0.18914$ m.

The present study was performed using a series of multiblock structured computational grids prepared by the Royal Netherlands Aerospace Centre (NLR) as part of the AIAA 7th Drag Prediction Workshop. The provided grids take into account the aeroelastic deformations of the wing for each angle of attack. The geometries considered in this paper correspond to $\alpha = 2.5°$, 3°, 3.5°, and 4°. The outer size of the computational domain was 150 m. The computational grid contains 1408 blocks, 36.3 million cells and correctly describes model geometry and flow around the model. There were 112 cells along the aircraft wingspan, 64 cells along the wing chord, 420 cells along the length of the fuselage (Fig. 7). In order to simulate the boundary layer correctly, 32 cells were placed in the near-wall layer of blocks, the dimensionless parameter of the first cell height near the solid surface y^+ did not exceed 1 for the whole model.

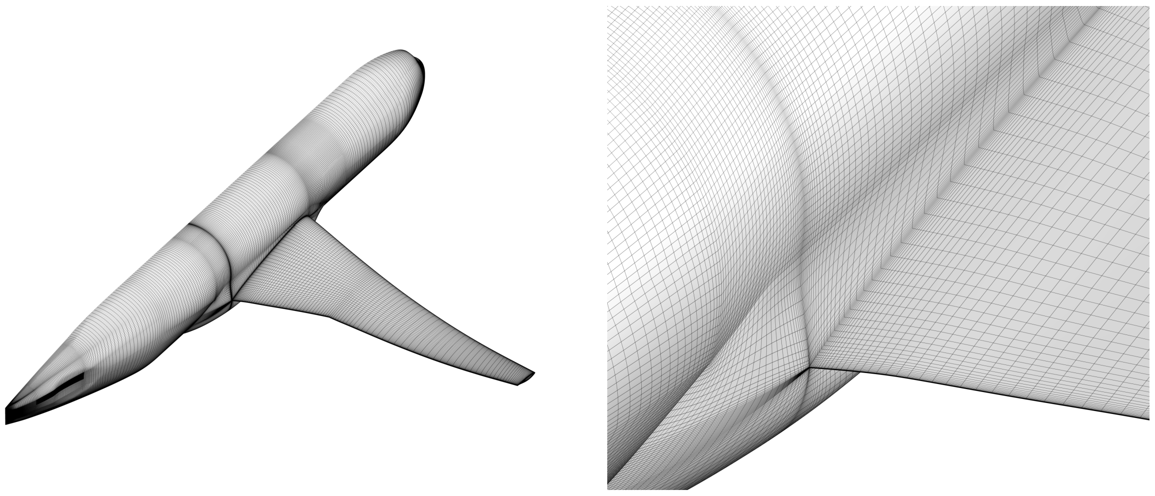


Figure 7. CRM surface grids

It is necessary to note the following observation made during work with the NLR series of grids. The linear dimensions of the model (wingspan, fuselage length, etc.), for which the grids were constructed, are smaller than the dimensions of the original CRM model by about 1.06 times. Therefore, all results presented in this paper were obtained on the grids scaled up by a factor of 1.06.

Numerical investigation was performed at the Reynolds number $\mathrm{Re}_c = 20 \cdot 10^6$ and Mach number $\mathrm{M} = 0.85$. Six angles of attack were specified between $2.5°$ and $5.0°$ deg at $0.5°$ increments. For simulations with angles of attack $\alpha = 4.5°$ and $5°$, mesh on deformed geometry for angle $\alpha = 4°$ was used. The results obtained with the original SST model and its non-linear modification SST NL are compared.

Figure 8 shows isentropic Mach number fields and streamlines on the upper wing surface at the angle of attack $\alpha = 5°$. It is evident that the obtained flow patterns differ significantly. The main difference is associated with the side of body (SOB) separation in the angular flow at the junction of the wing and fuselage. This separation appears in the SST calculations at $\alpha = 4°$ and grows as the angle of attack increases. In the results obtained using the SST NL model, SOB separation does not appear.

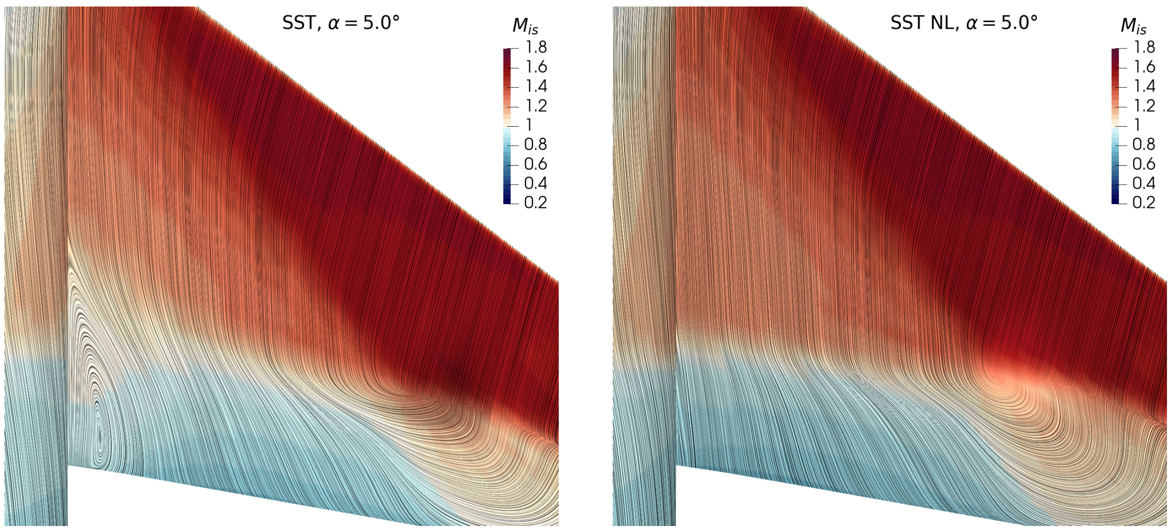


Figure 8. Isoentropic Mach number and streamlines on the model surface, $\mathrm{M} = 0.85,\ \alpha = 5.0°$

Another significant effect introduced by the nonlinear SST NL model is the upstream shift of the shock wave. Given at the Fig. 9 difference of the pressure coefficient values ($\Delta C_p = C_p^{\mathrm{nl}} - C_p^{\mathrm{sst}}$) illustrates the variations in the flow over the wing. At the angle of attack $3°$, there is no SOB separation and the only difference in the flow is associated with a slight shift of the shock wave. As the angle of attack increases, the differences become more significant: in the SST solution SOB separation appears; in the SST NL solution shock wave shift increases. As a result, these differences lead to discrepancy in the aerodynamic characteristics, in particular, the lift coefficient C_L (Fig. 10). As can be seen in the figure, the discrepancy in the values of C_L is approximately 0.02.

Conclusion

The paper presents the results of simulation of anisotropic turbulent flows using the nonlinear model SST NL. Three tests are considered: flow in a square duct, corner flow separation at a

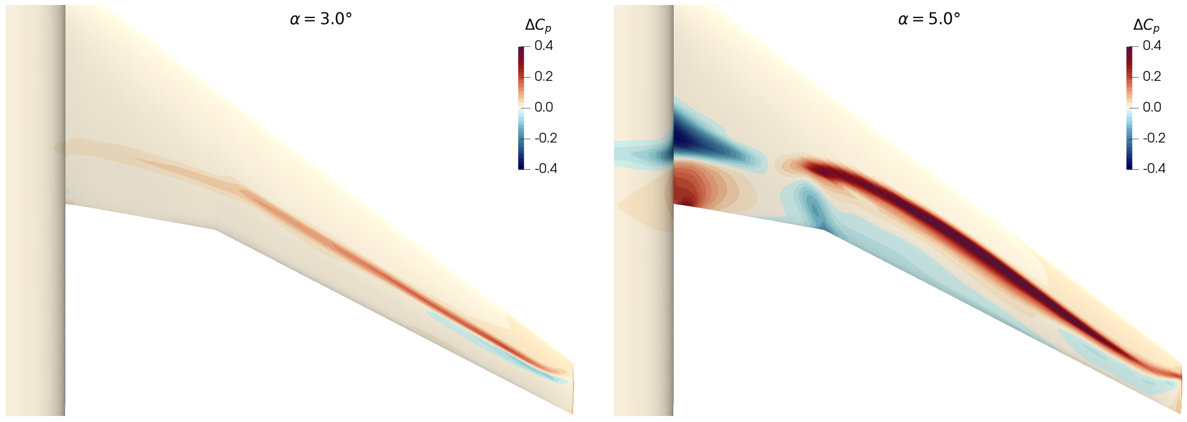


Figure 9. Difference in pressure coefficient values $\Delta C_p = C_p^{\mathrm{nl}} - C_p^{\mathrm{sst}}$, $\alpha = 3°$ and $5°$

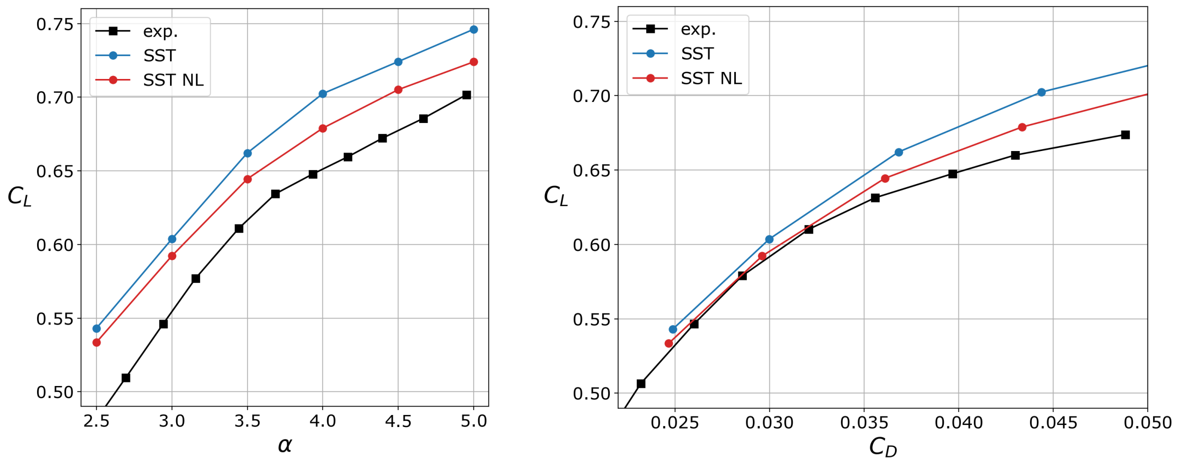


Figure 10. CRM wing-body lift and drag polar, M = 0.85, Re = $20 \cdot 10^6$ (experimental data [22])

wing-body junction, and transonic wing flow. The square duct flow calculations show that, in contrast to the linear SST, the SST NL model can simulate the secondary flow and therefore provides a significant increase in the accuracy of the main flow calculation. The test with juncture flow shows that the SST model tends to overestimate the separation size, which is typical for linear eddy viscosity models. On the contrary, the SST NL model correctly predicts the separation length, but underestimates its width. The results of the test with transonic wing flow do not allow drawing an unambiguous conclusion. On the one hand, the nonlinear model makes it possible to eliminate the false corner flow separation at the wing-body junction and thus to achieve a qualitative improvement in the simulation results. On the other hand, the nonlinear model predicts an upstream shifted location of the shock wave on the upper surface of the wing, which leads to underestimation of the lift force. This problem requires further investigation, possibly with a subsequent recalibration of the nonlinear term of the model.

Nevertheless, it can be concluded that the nonlinear model SST NL allows to significantly improve the quality of anisotropic flows simulation compared to linear SST model. Moreover, the SST NL model does not require a significant increase in computational cost as compared to the conventional SST. Tests have shown that the number of iterations required for convergence does not change, while the computation time of one iteration increases by 4–5 %.

Acknowledgements

This work is supported by the Russian Science Foundation under grant No. 22-29-00660, https://rscf.ru/project/22-29-00660/.

References

1. Allmaras, S.R., Johnson, F.T., Spalart, P.R.: Modifications and clarifications for the implementation of the Spalart–Allmaras turbulence model. In: Seventh International Conference on Computational Fluid Dynamics (ICCFD7). pp. 1–11. No. 1902 (2012)

2. Bosnyakov, S., Kursakov, I., Lysenkov, A., *et al.*: Computational tools for supporting the testing of civil aircraft configurations in wind tunnels. Progress in Aerospace Sciences 44(2), 67–120 (2008). `https://doi.org/10.1016/j.paerosci.2007.10.003`

3. Bosnyakov, S.M., Gorbushin, A.R., Kursakov, I.A., *et al.*: About verification and validation of computational methods and codes on the basis of Godunov method. TsAGI Science Journal 48(7), 597–615 (2017). `https://doi.org/10.1615/TsAGISciJ.2018026173`

4. Cecora, R.D., Radespiel, R., Eisfeld, B., Probst, A.: Differential Reynolds-stress modeling for aeronautics. AIAA Journal 53(3), 739–755 (2015). `https://doi.org/10.2514/1.J053250`

5. Garbaruk, A.V.: Numerical simulation and stability analysis of wall-bounded turbulent flows. DSc Thesis, Peter the Great St. Petersburg Polytechnic University (2020)

6. Kegerise, M.A., Neuhart, D.H.: An experimental investigation of a wing-fuselage junction model in the NASA Langley 14- by 22-foot subsonic tunnel. NASA TM (220286), 1–195 (2019). `https://doi.org/10.2514/6.2019-0077`

7. Launder, B.E., Reece, G.J., Rodi, W.: Progress in the development of a Reynolds-stress turbulence closure. Journal of Fluid Mechanics 68(3), 537–566 (1975)

8. Matyash, E.S., Savelyev, A.A., Troshin, A.I., Ustinov, M.V.: Allowance for gas compressibility in the γ-model of the laminar-turbulent transition. Computational Mathematics and Mathematical Physics 59(10), 1720–1731 (2019). `https://doi.org/10.1134/S0965542519100117`

9. Matyushenko, A.A., Garbaruk, A.V.: Non-linear correction for the k-ω SST turbulence model. Journal of Physics: Conference Series 929(1), 1–6 (2017). `https://doi.org/10.1088/1742-6596/929/1/012102`

10. Menter, F.R., Garbaruk, A.V., Egorov, Y.: Explicit algebraic Reynolds stress models for anisotropic wall-bounded flows. Progress in Flight Physics 3, 89–104 (2012). `https://doi.org/10.1051/eucass/201203089`

11. Menter, F.R., Kuntz, M., Langtry, R.: Ten years of industrial experience with the SST turbulence model. Turbulence Heat and Mass Transfer 4, 625–632 (2003)

12. Menter, F.R.: Zonal two equation k-ω turbulence models for aerodynamic Flows. AIAA Paper (2906) (1993)

13. Morrison, J.H.: 7th AIAA CFD Drag Prediction Workshop. `https://aiaa-dpw.larc.nasa.gov`, accessed: 2022-09-10

14. Morrison, J.H., Kleb, B.: Observations on CFD verification and validation from the AIAA drag prediction workshops. AIAA Paper (0202), 1–21 (2014). `https://doi.org/10.2514/6.2014-0202`

15. Pope, S.: A more general effective-viscosity hypothesis. Journal of Fluid Mechanics 72, 331–340 (1975)

16. Raiesi, H., Piomelli, U., Pollard, A.: Evaluation of turbulence models using direct numerical and large-eddy simulation data. Journal of Fluids Engineering 133(2) (2011). `https://doi.org/10.1115/1.4003425`

17. Rivers, M.: NASA Common Research Model. `https://commonresearchmodel.larc.nasa.gov`, accessed: 2022-09-10

18. Rumsey, C.L., Lee, H.C., Pulliam, T.H.: Reynolds-averaged Navier–Stokes computations of the NASA Juncture Flow model using FUN3D and OVERFLOW. AIAA Paper (1304), 1–31 (2020). `https://doi.org/10.2514/6.2020-1304`

19. Rumsey, C.: NASA Juncture Flow. `https://turbmodels.larc.nasa.gov/Other_exp_Data/junctureflow_exp.html`, accessed: 2022-09-10

20. Spalart, P.R.: Strategies for turbulence modelling and simulations. International Journal of Heat and Fluid Flow 21(3), 252–263 (2000). `https://doi.org/10.1016/S0142-727X(00)00007-2`

21. Spalart, P.R., Allmaras, S.R.: A one-equation turbulence model for aerodynamic flows. AIAA Paper (0439) (1992). `https://doi.org/10.2514/6.1992-439`

22. Tinoco, E., Keye, S.: Drag predictions at and beyond cruise for the Common Research Model by an international collaborative community (2022)

23. Tinoco, E.N., Brodersen, O.P., Keye, S., *et al.*: Summary data from the sixth AIAA CFD drag prediction workshop: CRM cases. Journal of Aircraft 55(4), 1352–1379 (2018). `https://doi.org/10.2514/1.C034409`

24. Wallin, S., Johansson, A.V.: An explicit algebraic Reynolds stress model for incompressible and compressible turbulent flows. Journal of Fluid Mechanics 403, 89–132 (2000). `https://doi.org/10.1017/S0022112099007004`

DOI: 10.14529/jsfi220405

Adapting a Scientific CFD Code to Industrial Applications on Hybrid Supercomputers

Andrey V. Gorobets[1]

The NOISEtte heterogeneous parallel code for simulating turbulent flow and aerodynamic noise is considered. In our previous works, high acceleration and parallel efficiency in scientific scale-resolving simulations using GPUs were reported. For parallelization, the MPI, OpenMP and OpenCL standards are used, the latter allows using GPUs from different vendors. However, the further transition to industrial-oriented applications brought more trouble. Instead of discussing the parallel algorithm, this work will focus on the problems that are not so obvious at first glance, which arise when developing a heterogeneous simulation code. How to deal with numerous simulation algorithm components, all those bells and whistles like wall functions, mixing plane and sliding interfaces, synthetic turbulence generators, a variety of boundary conditions, etc., that either need to be ported to the GPU side or incorporated directly from the CPU side? How to maintain and modify the OpenCL code in a growing number of source files? How to arrange the modularity of a complex heterogeneous software package? How to preserve reliability and fault tolerance, especially in the case of numerical schemes of increased accuracy, but reduced social responsibility? These issues are discussed here and some solutions will be proposed.

Keywords: heterogeneous code, computational fluid dynamics, turbulent flows, scale-resolving simulation, CPU+GPU, MPI+OpenMP+OpenCL.

Introduction

Computational fluid dynamics (CFD) software is one of the main burners of supercomputer time (as well as the associated electricity and taxpayers' money). Due to the high resource intensity, the efficient use of hybrid systems is critical, and GPU computing has long been widespread in CFD applications. CFD codes are constantly evolving in the use of hybrid supercomputers, improving scalability, enabling multi-GPU and heterogeneous computing capabilities. Many successful examples can be found in both scientific and commercial codes, see e.g. [1–3, 8] among many others, or specifications of commercial codes capable of multi-GPU computing, such as Simcenter STAR-CCM+, Ansys Fluent multi-GPU solver, GPU-optimized Altair software, including AcuSolve or its LBM-based flow solver ultraFluidX [6].

In the present work, a typical supercomputer time burner is considered, namely, the NOISEtte heterogeneous code for modeling turbulent flow and its aerodynamic noise. The parallel algorithm and performance on various GPU-based hybrid systems were already presented in detail in [4]. Here the focus will be not on the parallelization itself but on the problems that are not so obvious at first glance, which arise when developing a heterogeneous simulation code.

The article is organized as follows. Section 1 outlines numerical methods and algorithms. Section 2 is devoted to complexities of industrial applications. In Section 3, the modular architecture is presented. Section 4 is devoted to code reliability. Conclusions summarize the study.

1. Numerical and Parallel Framework

The turbulent flow of a viscous compressible gas is governed by the Navier–Stokes (NS) equations. The numerical algorithm is based on higher accuracy schemes on unstructured mixed-element meshes, hybrid RANS-LES approaches for turbulence modeling, implicit time integra-

[1]Keldysh Institute of Applied Mathematics, RAS Moscow, Russian Federation

tion. More information on our simulation technology and all its components can be found in [5] and references therein. For parallelization, the MPI, OpenMP and OpenCL standards are used, which allows us to cynically occupy an excessively large number of hybrid cluster nodes and to engage GPUs from different vendors. Two-level mesh partitioning is used for distribution of workload between cluster nodes and devices inside nodes. Overlapping computations and communications helps to achieve better parallel efficiency. Comprehensive information regarding the parallel algorithm is presented in [4]. The parallel performance demonstrated there includes CPU-only systems using up to about 10 thousand cores, GPU-based hybrid clusters using several dozen GPUs, obtaining the equivalent of about 150 to 200 CPU cores per GPU. The considered simple test cases were limited to an external flow on a static mesh.

2. Industrialization

When it comes to industrial problems, there are more components of the numerical methodology involved, for instance, wall functions, mixing plane and sliding interfaces, synthetic turbulence generators (STG) and sponge layers, a variety of boundary conditions (BC), deforming meshes, immersed boundary methods, more complex equations of state, among many others. Many of these components may seem insignificant as on CPUs they are responsible for a very minor fraction of the overall computing time, some even below 1%. Because of this "insignificance", there is a temptation to leave this functionality on the CPU side, since porting it to the GPU is no easier than the main parts of the simulation algorithm. But that does not work, since the GPU is an order of magnitude faster, hence the weight of the things left on the CPU becomes an order of magnitude bigger, respectively. The need to exchange extra data with the GPU enlarges this weight several times more. Thus, such small components, especially if there are many, can easily take up more than half of the computational time. It is easy to conclude that most of the effort has to be spent on such "insignificant" things in order to deal with industrial applications.

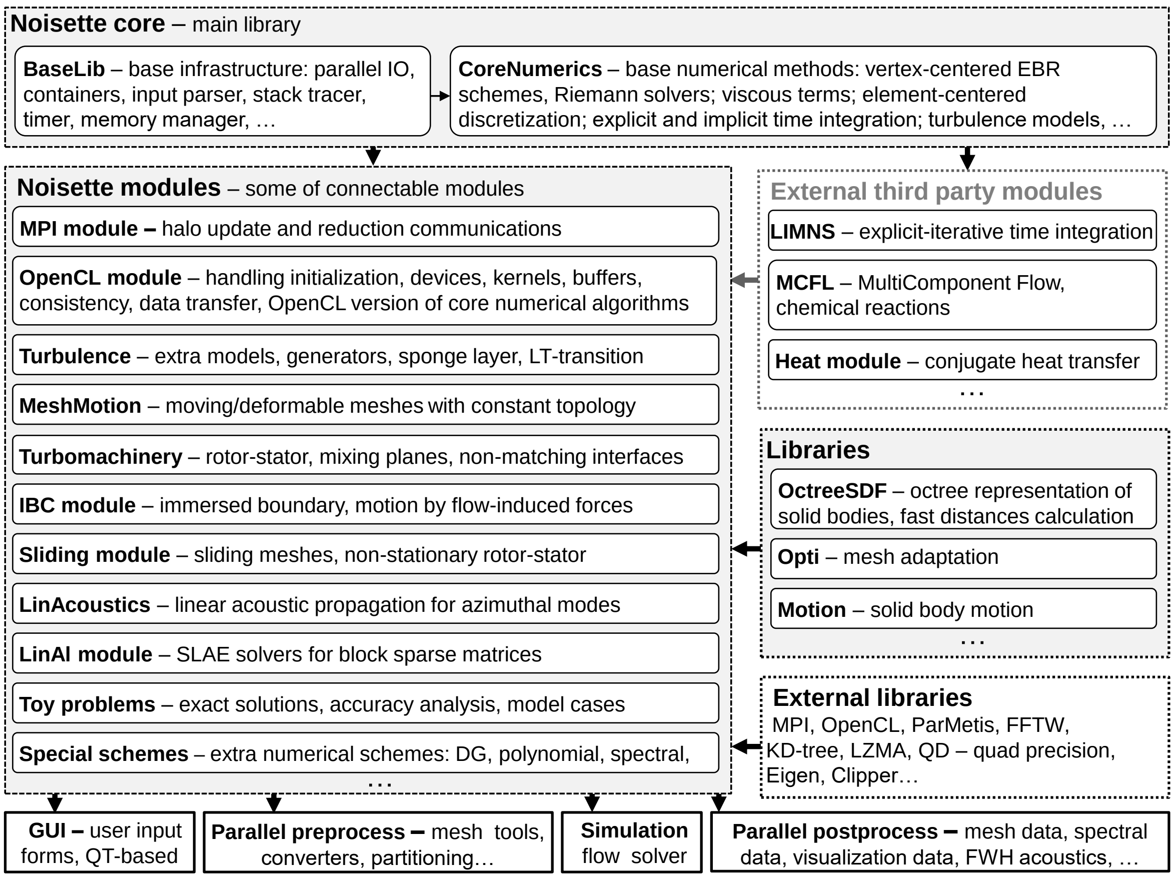
Porting so many things, in turn, imposes more problems related with maintainability and modifiability of so many OpenCL kernels. The OpenCL source code becomes organized in plenty of files. These files, when passed to the compiler at runtime of the CPU program, are concatenated with each other and with runtime-generated preprocessor definitions. This makes the compiler log hard to interpret, since the line numbers it reports are irrelevant. To solve this issue, the program module responsible for the OpenCL part tracks the number of lines in files and the assembly order of the source code, then it parses the compiler log and restores the actual files and line numbers, as explained in [4].

Apart from the code complexity, the time step in stationary RANS simulations typical of industrial applications is much bigger as it is not limited by the dynamics of turbulent structures in scale-resolving simulations. This, in turn, needs a more powerful linear solver for the Jacobi system in the implicit time integration. To mitigate this problem, our preconditioned BiCGSTAB solver had to be upgraded with slightly more complex preconditioners than the basic block Jacobi method. To prevent the solver from breaking down at large time steps, preconditioners based on the multicolor Gauss–Seidel (GS) method, typical for GPU computing, have been implemented in a heterogeneous way, combining MPI, OpenMP for CPUs and OpenCL for GPUs. This required notable additional effort, and to go further, it may be even necessary to use a multigrid approach and connect external solvers capable of working with sparse block matrices.

Finally, here are some particular examples. Boundary conditions used to take about 1–2% of the overall time when running on CPUs. When running on GPUs, leaving the BCs on the CPU side results in a cost of about 20%, where most of the time, around 3/4, is spent on the extra traffic between CPU and GPU devices (due to the need for transferring Jacobi matrix blocks). Porting the BCs to the GPU side, which is rather laborious, has reduced the cost to 2–3%, that is, an order of magnitude less. Similarly, on CPUs, the STG was consuming about 3%. Being too lazy to port it to the GPU also costs about 20%, where 1/4 goes to traffic, 3/4 to computational expenses. Porting the STG has shrinked its contribution to 4%. Regarding the linear solver upgrade, heterogeneous implementations of multicolor variants of GS-based preconditioners (GS, SGS, SOR, SSOR) demonstrate as high acceleration as for the block Jacobi method, which is about 8 times on NVIDIA V100 vs. 16-core Intel Xeon Gold.

3. Modularity

The ongoing expansion of functionality and complication of the code required a transition to a modular architecture. The code has been split into the core part and connectable modules and libraries, as shown in Fig. 1. The core contains the basic infrastructure and numerical methods.



Noisette core – main library
BaseLib – base infrastructure: parallel IO, containers, input parser, stack tracer, timer, memory manager, …
CoreNumerics – base numerical methods: vertex-centered EBR schemes, Riemann solvers; viscous terms; element-centered discretization; explicit and implicit time integration; turbulence models, …
Noisette modules – some of connectable modules
MPI module – halo update and reduction communications
OpenCL module – handling initialization, devices, kernels, buffers, consistency, data transfer, OpenCL version of core numerical algorithms
Turbulence – extra models, generators, sponge layer, LT-transition
MeshMotion – moving/deformable meshes with constant topology
Turbomachinery – rotor-stator, mixing planes, non-matching interfaces
IBC module – immersed boundary, motion by flow-induced forces
Sliding module – sliding meshes, non-stationary rotor-stator
LinAcoustics – linear acoustic propagation for azimuthal modes
LinAl module – SLAE solvers for block sparse matrices
Toy problems – exact solutions, accuracy analysis, model cases
Special schemes – extra numerical schemes: DG, polynomial, spectral,
…
External third party modules
LIMNS – explicit-iterative time integration
MCFL – MultiComponent Flow, chemical reactions
Heat module – conjugate heat transfer
…
Libraries
OctreeSDF – octree representation of solid bodies, fast distances calculation
Opti – mesh adaptation
Motion – solid body motion
…
External libraries
MPI, OpenCL, ParMetis, FFTW, KD-tree, LZMA, QD – quad precision, Eigen, Clipper…
GUI – user input forms, QT-based
Parallel preprocess – mesh tools, converters, partitioning…
Simulation flow solver
Parallel postprocess – mesh data, spectral data, visualization data, FWH acoustics, …

Figure 1. Modular structure of the simulation code

The infrastructure includes such things as parallel IO, containers for mesh data, user input parser, internal memory manager, stack tracer, timer, etc. The computing core contains basic numerical methods: the convective part of the NS equations, including vertex-centered EBR schemes and its cell-centered variants, Riemann solvers, low-Mach preconditioner; methods for calculating the viscous terms; explicit and implicit time integration; basic RANS and LES turbulence models, etc. Modules contain extra functionality that works on a base of the core. Libraries, in contrast to modules, do not use the core. The code can be built without any library or module (even without MPI, OpenMP and OpenCL). Simply removing the source code folder of a module or library eliminates all of its functionality from the code. The build system automatically adapts to the available code configuration and sets the necessary definitions.

In the core simulation algorithm, functions are equipped wherever needed with connection points simply on a base of function pointers, allowing modules to override a function with its own implementation or to add its function calls into a function. If an operation has different options, such as different reconstructions, various Riemann solvers, etc., multiple-choice switches select the option defined by the user input. Such switches can be extended with more options using a special template class that stores more options, each given by a function pointer and a corresponding text label for user input (only the one selected by the user is active). Thus, modules can add their functions wherever needed, override basic ones, add more options, add its data to checkpoint records, visualization etc. For ideas on how this works, see Fig. 2. It appeared, that this very simple approach with function pointers allows to implement rather complex things without introducing changes into the core, such as multi component flows, chemical reactions, conjugated heat transfer, etc.

```
// Connection points to some function
typedef void (*tSomeFunc)(); // function pointer of some particular type
tSomeFunc ExtVersion=NULL; // external version of some function
vector<tSomeFunc> ExtActions; // external actions inside some function
void OverrideSomeFunction(tSomeFunc f){ ExtVersion = f; }
void AddActionToSomeFunction(tSomeFunc f){ ExtActions.push_back(f); }

void SomeFunction(){
  if(ExtVersion) return ExtVersion(); // replacement with an external version
  SomeBasicFunctionality();
  for(size_t i=0; i<ExtActions.size(); ++i) ExtActions[i](); // external actions
}

// Extensible implementation options
enum tBaseEnum{ BaseOption1 = 1, BaseOption2 = 2 };
tExtEnum<tBaseEnum, tSomeFunc> SomeChoice; // extensible enum wrapper
void SomethingWithVariousOptions(){
  switch((tBaseEnum)SomeChoice){
  case BaseOption1: return BaseFunction1(); // one of basic options
  case BaseOption2: return BaseFunction2(); // one of basic options
  default: // one of registered external options chosen in user input
    return SomeChoice.GetExtOption();
  }
}

// Registering module’s functionality at initialization
void MyModuleInit(){
  OverrideSomeFunction(MyVersionOfSomeFunction);
  SomeChoice.RegisterOption(MyOptionForSomeChoice1, “myoption1”);
  SomeChoice.RegisterOption(MyOptionForSomeChoice2, “myoption2”);
}
```

Figure 2. An illustration of how connecting a module works

The last thing regarding modules is how to manage what is implemented for the GPU, what can be incorporated from the CPU, and what is not available for GPU computing. A special function checks this compatibility of the actual code configuration with the given user input and available implementations. If a module is requested in a GPU-enabled execution, and there is no GPU implementation available, the execution aborts with the information on what is missing and what should be changed to make it run. Note that it also takes some extra effort to implement and maintain such a compatibility check.

4. Reliability

Implementation consistency across different architectures is critical to heterogeneous software reliability. Additional measures, rather laborious, have been taken to protect the code from inconsistencies and errors. Such measures include expanding the quality assurance suite with tests covering all GPU-enabled features and implementing internal consistency checks inside the code. Internal checks are performed each time a GPU-enabled simulation is started by running several time integration steps on both the CPU and GPU. First, per-kernel CPU vs GPU version check for each OpenCL kernel involved in a particular numerical algorithm ensures that its CPU counterpart produces the same results (with some specified round-off tolerance). Second, a full time step check ensures that the whole time step algorithms are consistent by comparing mesh function after several time steps.

Another important issue is the fault tolerance of the simulation algorithm itself, regardless of the devices on which it is executed. Non-physical flow instabilities can appear due to insufficient mesh quality or resolution in the case of a low-dissipation scheme, too big time step size, insufficient linear solver accuracy, etc. To prevent simulation breakdown, the flow fields are checked every certain number of time steps to detect problems such as too high or too low density or pressure (these limits are case specific and are set by user depending e.g. on the actual Mach number), incorrect numbers, etc. In the case if flow fields turn out to be incorrect, the simulation automatically returns to the previously stored in memory restart checkpoint and adjusts the relevant parameters such as the time step size, solver tolerance, upwind and central difference weights, etc. When GPU computing is enabled, such checkpoints are stored in CPU memory to save scarce GPU memory. This requires additional transfers of mesh functions and processing routines in the GPU-enabled algorithm.

Conclusions

In CFD applications, GPUs increase performance by an order of magnitude compared to an equivalent number of CPUs. For this, the simulation algorithm must be adapted to the more restricted parallel paradigm used on GPUs. Reducing memory consumption is also critical for GPU computing. The underlying computational algorithm then needs to be efficiently implemented for the GPU architecture. But besides these obvious things, it turned out that there are still many problems that need to be solved, and a lot of work that needs to be done in order to effectively use a heterogeneous code in practice. This short communication provides a summary of such problems and how to solve them. A significant expansion of functionality towards industrial applications required a transition to a modular architecture. Porting additional components of the simulation algorithm to OpenCL, even those that take a negligible amount of computational time when running on the CPU, appeared to be unavoidable. Otherwise, the performance would be about twice as low. The growing amount of OpenCL code required additional measures to improve reliability. The need for stationary RANS simulations required the linear solver upgrade in the implicit time integration scheme. Eventually, the code seems to have become applicable for stationary and scale-resolving CFD simulations, including cases with synthetic turbulence; mixing-plane rotor-stator interfaces; various boundary conditions for solid surfaces, inflow, outflow; rotating coordinate system; various turbulence modeling approaches and models, etc. Practical performance on GPUs gives us the equivalent of 100 to 200 CPU cores per device, which is well worth the effort.

Acknowledgements

This work was funded by the RSF project No. 19-11-00299. The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University [7], the equipment of Shared Resource Center of KIAM RAS (http://ckp.kiam.ru), the infrastructure of the Shared Research Facilities "High Performance Computing and Big Data" (CKP "Informatics") of FRC CSC RAS (Moscow). The author thankfully acknowledges these institutions.

References

1. Alvarez, X., Gorobets, A., Trias, F., *et al.*: HPC2 – A fully-portable, algebra-based framework for heterogeneous computing. Application to CFD. Computers & Fluids 173, 285–292 (2018), `https://doi.org/10.1016/j.compfluid.2018.01.034`

2. Bocharov, A., Evstigneev, N., Petrovskiy, V., *et al.*: Implicit method for the solution of supersonic and hypersonic 3D flow problems with Lower-Upper Symmetric-Gauss-Seidel preconditioner on multiple graphics processing units. Journal of Computational Physics 406, 109189 (2020), `https://doi.org/10.1016/j.jcp.2019.109189`

3. Borrell, R., Dosimont, D., Garcia-Gasulla, M., *et al.*: Heterogeneous CPU/GPU co-execution of CFD simulations on the POWER9 architecture: Application to airplane aerodynamics. Future Generation Computer Systems 107, 31–48 (2020), `https://doi.org/10.1016/j.future.2020.01.045`

4. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271, 108231 (2022), `https://doi.org/10.1016/j.cpc.2021.108231`

5. Gorobets, A., Duben, A.: Technology for Supercomputer Simulation of Turbulent Flows in the Good New Days of Exascale Computing. Supercomputing Frontiers and Innovation 8(4), 4–10 (2021), `https://doi.org/10.14529/jsfi210401`

6. Niedermeier, C., Janssen, C., Indinger, T.: Massively-parallel multi-GPU simulations for fast and accurate automotive aerodynamics. In: Proceedings of the 7th European Conference on Computational Fluid Dynamics, Glasgow, Scotland, UK, June 11–15, 2018 (06 2018)

7. Voevodin, V., Antonov, A., Nikitenko, D., *et al.*: Supercomputer Lomonosov-2: Large Scale, Deep Monitoring and Fine Analytics for the User Community. Supercomput. Front. Innov. 6(2), 4–11 (2019), `https://doi.org/10.14529/jsfi190201`

8. Watanabe, S., Aoki, T.: Large-scale flow simulations using lattice Boltzmann method with AMR following free-surface on multiple GPUs. Computer Physics Communications 264, 107871 (2021), `https://doi.org/10.1016/j.cpc.2021.107871`

DOI: 10.14529/jsfi220406

Penalized Wall Function Method for Turbulent Flow Modeling

Natalia S. Zhdanova[1], *Oleg V. Vasilyev*[1]

A novel penalized wall function method for simulations of wall-bounded compressible turbulent flows is proposed. The new approach is based on the Reynolds-averaged Navier–Stokes (RANS) equations to model the outer region of the turbulent boundary layer, while the inner part is approximated by the equilibrium wall function model. The differential formulation to match the external and the wall function solutions is reformulated in a form of the generalized characteristic-based volume penalization method to model the transfer of the shear stress from the outer region of the boundary layer to the wall and to impose the wall-stress boundary conditions on the RANS solution. The exchange location is specified implicitly through a localized source term in the boundary layer equation, written as a function of the normalized distance from the wall. The wall-stress condition is determined by solving an auxiliary equation for the wall-stress, ensuring the correct matching of the RANS and the wall function solutions at the exchange layer. The proposed method noticeably reduces the near-wall mesh resolution requirements without significant modification of the RANS solver and removes the ill-defined explicit matching procedure, commonly used by traditional wall function-based methods. The penalized wall function approach is implemented using the vertex-centered control volume method on unstructured computational grids. The effectiveness of the developed penalized wall function method is demonstrated for two-dimensional bump-in-channel flow for the Spalart–Allmaras turbulence model.

Keywords: turbulence modeling, wall function, wall-bounded compressible turbulent flow, volume penalization.

Introduction

Accurate modeling of turbulent flows of engineering interest remains to be one of the major challenges of computational fluid dynamics. Due to prohibitively expensive computational cost of the direct numerical simulations (DNS) of large Reynolds number turbulent flows [24], a number of lower fidelity eddy-resolving approaches are currently actively being pursued. These methods are based either on the Reynolds-averaged Navier–Stokes (RANS) equations [14, 16, 31, 35] or on hybrid approaches [15, 36], in which the flow in the near-wall region is simulated using RANS-type models, while in regions far from the walls the Large Eddy Simulation (LES) method is used. The hybrid class of methods also includes detached eddy simulation (DES) methods [30, 32, 33] with a smooth transition from RANS to LES solutions.

Despite relatively moderate near-wall resolution requirements of RANS, hybrid RANS-LES, and DES approaches, these requirements are still significant and impose strict limitations on the computational resources, considerably increase the computation time, and complicate the computational grid construction.

Mesh resolution restrictions can be substantially reduced if the boundary solution is approximated by a wall function and only external RANS solution is simulated [11, 27]. This can be achieved by replacing no-slip wall boundary conditions with off-the-wall boundary conditions at the exchange location away from the wall. Alternatively a weak wall function formulation can be used to transfer the shear stress from the outer region of the boundary layer to the wall and to impose the wall-stress boundary conditions on the RANS solution [5, 13]. A weak wall function formulation is more preferable from the computational point of view due to its flexibility, but it does not guarantee an exact correspondence of the boundary layer displacement thicknesses,

[1]Keldysh Institute of Applied Mathematics, Russian Academy of Sciences, Moscow, Russian Federation

mainly due to approximate nature of the solution in the near-wall region with the slip velocity and turbulent viscosity extrapolated from the outer region of the solution, which results in a decrease of the displacement thickness compared to the RANS solution with no-slip boundary conditions.

In traditional wall function approaches [25] the boundary conditions are determined by solving nonlinear equations at the matching (exchange) location, which is not known a priori and is implicitly defined by the normalized distance from the wall, which, in turn, is a function of the wall shear stress. For this purpose, the solution at the exchange location, is interpolated from the nearest computational mesh points [7]. It should be noted that wall functions can be used in conjunction with immersed boundary methods, where additional constraints at the immersed mesh points are imposed to ensure the boundary conditions on the surface of the obstacle [6, 10, 12].

The main idea of the proposed approach, hereinafter referred to as the *Penalized Wall Function* (PWF) method, is to replace the nonlinear algebraic matching condition between the external and wall function solutions by a differential formulation based on a generalized characteristic-based volume penalization method [8, 19] to transfer the shear stress from the outer region of the boundary layer to the wall, while specifying exchange locations implicitly through the localized source term in the boundary layer equation, written as a function of the normalized distance from the wall. Such an approach, makes it possible to completely eliminate the need to explicitly find the exchange location, and, more importantly, reduces the system of partial differential equations with nonlinear algebraic constraints to a system of partial differential equations with differential feedback loop provided by characteristic penalty functions. The latter property makes it feasible to generalize the approach to problems with flow separation based on differential equilibrium [7, 20] and non-equilibrium [21, 26] wall functions. In general, the developed approach noticeably reduces the near-wall mesh resolution requirements without significant modification of the RANS solver. Note that the developed PWF method should be distinguished from hybrid approaches utilizing immersed boundary methods to set boundary conditions on the surface of the obstacles [6, 10], since it uses characteristic-based volume penalization to match external and wall function solutions. Furthermore, the PWF method can also be generalized for complex geometry flows, based on the already developed volume penalization methods [2, 8, 19, 22, 23, 37].

The rest of the paper is organized as follows. In Section 1 the governing equations of the numerical simulations, including the Favre-averaged Navier–Stokes equations for compressible flows and the evolution equations for turbulence models are introduced. The penalized wall function method for turbulent flow modeling is formulated in Section 2. The numerical method used to implement the PWF approach is briefly described in Section 3. The 2D bump-in-channel flow configuration and the corresponding simulation results using the PWF method are presented and discussed in Section 4. Concluding remarks are given in Section 4.1.

1. Governing Equations

1.1. Favre-averaged Navier–Stokes Equations

Compressible RANS equations are formulated in terms of Reynolds-averaged and Favre-averaged dependent variables. Denoting Reynolds-averaging and Favre-averaging operations as $\langle\phi\rangle$ and $\{\phi\} = \langle\rho\phi\rangle/\langle\rho\rangle$, respectively, where ϕ stands for a generic physical variable, the method involves Reynolds-averaged density $\langle\rho\rangle$ and pressure $\langle p\rangle$, together with Favre-averaged velocity $\{u_i\}$, temperature $\{T\}$, and total energy per unit mass $\{e\}$. For the sake of clarity, the corresponding operator symbols hereafter are omitted in the notations of Reynolds/Favre-averaged primitive variables and simple symbols ρ, p, u_i, T and e are used hereafter to denote the Reynolds-averaged density, pressure, and Favre-averaged velocity, temperature, and total energy per unit mass. Therefore, the Favre-averaged Navier–Stokes equations for the conservation of mass, momentum, and energy in compressible flows for calorically perfect gas after incorporation of modeled turbulent terms using the Boussinesq eddy viscosity, eddy-conductivity, and constant turbulent Prandtl number assumptions can be written in the following general form:

$$\begin{aligned}
\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u_j)}{\partial x_j} &= 0, && (1)\\
\frac{\partial \rho u_i}{\partial t} + \frac{\partial}{\partial x_j}(\rho u_i u_j) &= -\frac{\partial p}{\partial x_i} + \frac{\partial \hat{\tau}_{ij}}{\partial x_j}, && (2)\\
\frac{\partial \rho e}{\partial t} + \frac{\partial}{\partial x_j}\left[(\rho e + p)\, u_j\right] &= \frac{\partial}{\partial x_j}\left[u_i \hat{\tau}_{ij} - q_j\right], && (3)
\end{aligned}$$

where

$$\begin{aligned}
p &= \rho R T, && (4)\\
e &= \frac{1}{2} u_i u_i + \frac{p}{\rho(\gamma - 1)}, && (5)\\
q_j &= -c_p \left(\frac{\mu}{Pr_{\mathrm{L}}} + \frac{\mu_{\mathrm{T}}}{Pr_{\mathrm{T}}}\right) \frac{\partial T}{\partial x_j}, && (6)\\
\hat{\tau}_{ij} &= 2\mu \tilde{S}_{ij} + \tau_{ij}, &&\\
\tau_{ij} &= 2\mu_{\mathrm{T}} \tilde{S}_{ij}, && (7)\\
\tilde{S}_{ij} &= \operatorname{dev}(S_{ij}) = S_{ij} - \frac{1}{3}\frac{\partial u_k}{\partial x_k}\delta_{ij}, &&\\
S_{ij} &= \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right). &&
\end{aligned}$$

In the equations above, the parameter R stands for the gas constant, while c_v and c_p are the specific heat constants at constant volume and pressure, respectively. The specific heat ratio $\gamma = \frac{c_p}{c_v} \equiv 1.4$ for diatomic gases is assumed. The term q_j is the sum of both the laminar and the modeled turbulent heat flux vectors, with $Pr_{\mathrm{L}} = 0.72$ and $Pr_{\mathrm{T}} = 0.9$ being the laminar and the turbulent Prandtl numbers, respectively. The turbulent eddy viscosity is denoted by μ_{T}, which is unknown and needs turbulence models for closure. The term $\hat{\tau}_{ij}$ is the sum of the molecular and the Reynolds stress tensors, while τ_{ij} is the Reynolds stress tensor, S_{ij} is the mean strain-rate tensor, and $\tilde{S}_{ij}$ is the deviatoric part of S_{ij}. For simplicity of consideration, constant dynamic molecular viscosity μ is assumed, which is a good approximation for low Mach number flows considered in this paper.

1.2. Turbulence Model Equations

Without loss of generality, the Spalart–Allmaras (S-A) turbulence model [31] is used to illustrate the developed penalized wall function approach. The S-A model is widely used as a turbulence model closure for the equations (6) and (7), including high-velocity flows with a significant effect of compressibility [4].

The standard Spalart–Allmaras model [31] in terms of $\rho\tilde{\nu}$ can be written as follows:

$$\begin{aligned}\frac{\partial\rho\tilde{\nu}}{\partial t}+\frac{\partial}{\partial x_j}\left(\rho\tilde{\nu}u_j\right)&=c_{b1}(1-f_{t2})\tilde{S}\rho\tilde{\nu}-\left[c_{w1}f_w-\frac{c_{b1}}{\kappa^2}f_{t2}\right]\rho\left(\frac{\tilde{\nu}}{\delta}\right)^2\\&+\frac{\partial}{\partial x_j}\left[\left(\frac{\mu}{\sigma}+\frac{\rho\tilde{\nu}}{\sigma}\right)\frac{\partial\tilde{\nu}}{\partial x_j}\right]-\left(\frac{\mu}{\sigma\rho}+\frac{\tilde{\nu}}{\sigma}\right)\frac{\partial\rho}{\partial x_j}\frac{\partial\tilde{\nu}}{\partial x_j}+c_{b2}\frac{\rho}{\sigma}\frac{\partial\tilde{\nu}}{\partial x_j}\frac{\partial\tilde{\nu}}{\partial x_j},\end{aligned}\tag{8}$$

where the eddy viscosity is computed by

$$\mu_{\mathrm{T}}=\rho\tilde{\nu}f_{v1},\tag{9}$$

and auxiliary variables are defined as:

$$\begin{aligned}f_{v1}&=\frac{\chi^3}{\chi^3+c_{v1}^3},\\\chi&=\tilde{\nu}/\nu,\\\tilde{S}&=\max\left[0.3\sqrt{2\Omega_{ij}\Omega_{ij}},\sqrt{2\Omega_{ij}\Omega_{ij}}+\frac{\tilde{\nu}}{\kappa^2\delta^2}f_{v2}\right],\qquad(10)\\\Omega_{ij}&=\frac{1}{2}\left(\frac{\partial u_i}{\partial x_j}-\frac{\partial u_j}{\partial x_i}\right),\\f_{v2}&=1-\frac{\chi}{1+\chi f_{v1}},\\f_w&=g\left[\frac{1+c_{w3}^6}{g^6+c_{w3}^6}\right]^{1/6},\\g&=r+c_{w2}(r^6-r),\\r&=\min\left[\frac{\tilde{\nu}}{\tilde{S}\kappa^2\delta^2},10\right],\qquad(11)\\f_{t2}&=c_{t3}\exp(-c_{t4}\chi^2),\end{aligned}$$

where ν is kinematic molecular viscosity, $\delta(\mathbf{x})$ is the distance from the field point to the nearest wall, and to improve the stability of calculations the variable $\tilde{S}$ is bounded from below by the quantity $0.3\sqrt{2\Omega_{ij}\Omega_{ij}}$. The constant coefficients are prescribed as $c_{b1}=0.1355$, $c_{b2}=0.622$, $\sigma=2/3$, $\kappa=0.41$, $c_{w2}=0.3$, $c_{w3}=2$, $c_{v1}=7.1$, $c_{w1}=\frac{c_{b1}}{\kappa}+\frac{1+c_{b2}}{\sigma}$. Note that this "standard" version of the S-A model does not have the trip term "f_{t1}" and, hence, it is argued that f_{t2} is not necessary, i.e., $c_{t3}=0$ is assumed.

The following boundary condition on the wall surface

$$\tilde{\nu}=0\tag{12}$$

is used for consistency. When modeling external flows the constant turbulent viscosity $\tilde{\nu}=3\nu_\infty$ is assumed at the inflow boundary.

2. Penalized Wall Function Method

The considerably lower computational cost of eddy-resolving approaches compared to DNS or LES, makes RANS-based methods to be a method of choice for high Reynolds number turbulent flow simulations in aerospace industry. However, despite relatively moderate near-wall resolution requirements of RANS [14, 16, 31, 35], hybrid RANS-LES [15, 36], and DES [30, 32, 33] approaches, the direct resolution of flow structures of the RANS equations (1)–(3), (8) results in a considerable computational cost associated with a large number of mesh points in the near-wall region. Mesh resolution restrictions can be substantially reduced if instead of resolving the solution in the vicinity of the wall, it is approximated by a wall function and only external RANS solution is obtained [11, 27]. This can be achieved by replacing no-slip boundary conditions on the wall with the matching conditions between the wall function and the outer turbulent boundary layer solutions:

$$u_{\|}(\mathbf{x})\big|_{\delta(\mathbf{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}} = u_\tau f\left(\delta^+_{\mathrm{EL}}\right), \tag{13}$$

where the matching (exchange) location is determined from the following condition:

$$\delta(\mathbf{x}) = \frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}, \tag{14}$$

$u_{\|}(\mathbf{x})$ is the velocity component parallel to the surface at point $\mathbf{x}$, $\tilde{x}$ and $\tilde{y}$ are generalized coordinates along and normal to the wall, $\tilde{y}^+ = u_\tau\tilde{y}/\nu$ is the normalized coordinate, $u_\tau = \left(\tau_{\mathrm{w}}/\rho\right)^{1/2}$ is the friction velocity, τ_{w} is the wall stress, δ^+_{EL} is the normalized distance to the exchange location, and $f(\tilde{y}^+)$ is the wall function, defined as a function of the distance from the wall, normalized by the viscous length scale. Note that Eq. (14) can be viewed as a non-linear algebraic equation for determining the exchange location for a given coordinate $\tilde{x}$ on the wall. For the sake of simplicity, let us start by formulating the method for two-dimensional flows. The generalization of the method to three-dimensional flows will be provided at the end of the section.

For a given external velocity field $u_{\|}(\mathbf{x})$ and for a given normalized distance δ^+_{EL} the matching condition (13) is a non-linear algebraic equations for determining u_τ. A linear approximation of Eq. (13) for the friction velocity correction δu_τ can be written as

$$u_{\|}(\mathbf{x})\big|_{\delta(\mathbf{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}} + \left.\frac{\partial u_{\|}(\mathbf{x})}{\partial \mathbf{n}}\right|_{\delta(\mathbf{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}}\left(-\frac{\nu\delta^+_{\mathrm{EL}}}{u_\tau^2}\delta u_\tau\right) \approx u_\tau f\left(\delta^+_{\mathrm{EL}}\right) + \delta u_\tau f\left(\delta^+_{\mathrm{EL}}\right), \tag{15}$$

where the normal $\mathbf{n}$ is defined in terms of the distance function $\delta(\mathbf{x})$: $\mathbf{n} = \nabla\delta(\mathbf{x})$. Note that the second term on the left hand side of Eq. (15) arises due to the change of the distance $\delta(\mathbf{x})$ of the exchange location when u_τ changes and δ^+_{EL} is fixed. The solution of Eq. (15) for the friction velocity correction δu_τ is given by

$$\delta u_\tau \approx u_\tau \frac{u_{\|}(\mathbf{x})\big|_{\delta(\mathbf{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}} - u_\tau f\left(\delta^+_{\mathrm{EL}}\right)}{u_\tau f\left(\delta^+_{\mathrm{EL}}\right) + \left.\frac{\partial u_{\|}(\mathbf{x})}{\partial \mathbf{n}}\right|_{\delta(\mathbf{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}}\delta(\mathbf{x})}. \tag{16}$$

When a strong wall function formulation is used, the friction velocity correction (16) can be used in Newton's method to iteratively obtain the solution of the matching condition (13).

For the weak wall function formulation, the discrete friction velocity correction (16) can be replaced by temporal relaxation of the solution $u_\tau(\tilde{x}, t)$ on the time scale η_f at each point of the generalized coordinate $\tilde{x}$, defined along wall:

$$\frac{\partial u_\tau}{\partial t} = \frac{u_\tau}{\eta_f}\frac{u_{\|}(\mathbf{x})\big|_{\delta(\tilde{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}} - u_\tau f\left(\delta^+_{\mathrm{EL}}\right)}{u_\tau f\left(\delta^+_{\mathrm{EL}}\right) + \left.\frac{\partial u_{\|}(\mathbf{x})}{\partial \mathbf{n}}\right|_{\delta(\tilde{x})=\frac{\nu}{u_\tau}\delta^+_{\mathrm{EL}}}\delta(\tilde{x})}, \tag{17}$$

where for greater clarity the exchange location, implicitly defined by the equation (14), is explicitly written as $\delta(\tilde{x})$, while the no-slip boundary condition for the velocity $\mathbf{u}$ at the wall is replaced by the no-penetration condition for the normal velocity component $u_\perp(\mathbf{x}) = \mathbf{u}\cdot\mathbf{n}$:

$$u_\perp|_{\tilde{y}=0} = 0 \tag{18}$$

and the wall shear stress condition:

$$(\nu + \nu_\mathrm{T})\left.\frac{\partial u_\parallel}{\partial \mathbf{n}}\right|_{\tilde{y}=0} = u_\tau^2(\tilde{x}, t). \tag{19}$$

For consistent wall function formulation of the Spalart–Allmaras model, the turbulent viscosity from the outer flow region is transfered to the boundary, which can be easily achieved by replacing the distance function $\delta(\mathbf{x})$ in the Eqs. (8), (10) and (11) by

$$\delta(\mathbf{x}) = \max\left(\delta(\mathbf{x}), \frac{\nu}{u_\tau}\delta^+_\mathrm{EL}\right) \tag{20}$$

and changing the boundary condition (12) for turbulent viscosity to

$$\left.\frac{\partial \tilde{\nu}}{\partial \mathbf{n}}\right|_{\tilde{y}=0} = 0. \tag{21}$$

The equation (13) with an implicit determination of the exchange location $\delta(\tilde{x})$ complicates the solution of the problem, since, in general, exchange locations do not coincide with mesh points and the tangential velocity component $u_\parallel$ from the nearest mesh points needs to be interpolated to the exchange location, e.g., see [7]. The problem can be greatly simplified and the need to interpolate the tangential velocity to exchange locations can be completely eliminated by introducing an auxiliary friction velocity field $u_\tau(\mathbf{x}, t)$, defined in the entire domain and not only on the wall, and by replacing Eq. (17) for $u_\tau(\tilde{x}, t)$ on the wall by the following partial differential equation for the field $u_\tau(\mathbf{x}, t)$:

$$\frac{\partial u_\tau}{\partial t} - \underbrace{\mathcal{H}\left(\delta^+_\mathrm{EL} - \frac{u_\tau}{\nu}\delta(\mathbf{x})\right)\frac{l_s}{\eta_s}\frac{\partial u_\tau}{\partial \mathbf{n}}}_{\text{transfer of } u_\tau \text{ to the wall}} = \underbrace{\chi_\delta\left(\frac{\delta^+_\mathrm{EL} - \frac{u_\tau}{\nu}\delta(\mathbf{x})}{\sigma^+}\right)\frac{u_\tau}{\eta_f}\frac{u_\parallel(\mathbf{x}) - u_\tau f\left(\delta^+_\mathrm{EL}\right)}{u_\tau f\left(\delta^+_\mathrm{EL}\right) + \frac{\partial u_\parallel(\mathbf{x})}{\partial \mathbf{n}}\delta(\mathbf{x})}}_{\text{temporal relaxation of } u_\tau \text{ in an exchange layer}} + \underbrace{\nu_\mathrm{n}\Delta u_\tau}_{\text{smoothing}}, \tag{22}$$

where l_s and η_s are, respectively, the characteristic length and time scales of transferring the solution from the exchange layer to the wall, $\mathcal{H}(\xi)$ is the Heaviside function that disables the transfer of u_τ in the outer region of the turbulent boundary layer, $\chi_\delta(\xi)$ is a localized exchange layer masking function, for example, the Gaussian function

$$\chi_\delta(\xi) = \exp(-\xi^2/2), \tag{23}$$

σ^+ is the normalized thickness of the exchange layer, and $\nu_\mathrm{n}\Delta u_\tau$ is the numerical diffusion used to smooth out the auxiliary field u_τ. Note that in Eq. (22), the matching condition (13) is provided by temporal relaxation term in the spatially localized exchange layer, from which u_τ is transferred to the wall for the subsequent use in the boundary condition (19). The second term on the left hand side of Eq. (22) corresponds to the characteristic penalty function [8, 9, 19], which on the time scale η_s transfers the friction velocity to the wall.

In order to improve the convergence of the method in the earlier stages of the transient solution, the term $\frac{\partial u_{\|}(\mathbf{x})}{\partial \mathbf{n}}$ in the Eq. (22) can be approximated by differentiating the wall function solution:

$$\frac{\partial u_{\|}(\mathbf{x})}{\partial \mathbf{n}} \approx \frac{u_\tau^2}{\nu} f'\left(\frac{u_\tau \tilde{y}}{\nu}\right). \tag{24}$$

Substituting Eqs. (14) and (24) into Eq. (22) the following stabilized penalized wall function equation can be obtained:

$$\frac{\partial u_\tau}{\partial t} - \underbrace{\mathcal{H}\left(\delta^+_{\mathrm{EL}} - \frac{u_\tau}{\nu}\delta(\mathbf{x})\right)\frac{l_s}{\eta_s}\frac{\partial u_\tau}{\partial \mathbf{n}}}_{\text{transfer of } u_\tau \text{ to the wall}} = \frac{1}{\eta_f}\underbrace{\chi_\delta\left(\frac{\delta^+_{\mathrm{EL}} - \frac{u_\tau}{\nu}\delta(\mathbf{x})}{\sigma^+}\right)\frac{u_{\|}(\mathbf{x}) - u_\tau f\left(\delta^+_{\mathrm{EL}}\right)}{f\left(\delta^+_{\mathrm{EL}}\right) + f'\left(\delta^+_{\mathrm{EL}}\right)\delta^+_{\mathrm{EL}}}}_{\text{temporal relaxation of } u_\tau \text{ in an exchange layer}} + \underbrace{\nu_{\mathrm{n}}\Delta u_\tau}_{\text{smoothing}}. \tag{25}$$

The equation (25) is the basis of the penalized wall function method. Note that despite the fact that the developed approach has been demonstrated only in the context of the Spalart–Allmaras model, the PWF method can be used in conjunction with any turbulence model that allows transfer of shear stress from the outer region of the boundary layer to the wall. Note that the choice of the exchange location δ^+_{EL} can affect the accuracy and efficiency of the simulations. In general the exchange location should be chosen to be outside of the viscous sublayer region to improve the near-wall resolution requirements and below the law-of-the-wake region, which is problem dependent.

For three-dimensional flows, the problem can be rewritten in local two-dimensional coordinates, where the coordinate $\tilde{x}$ along the body surface corresponds to the direction of the tangential velocity component at the corresponding exchange location. In this case, the equation (25) can be used without modification under the assumption that the direction of the wall shear is aligned with the tangential velocity vector at the corresponding exchange location, and the variable $u_{\|}(\mathbf{x}) = \left\|\mathbf{u}_{\|}(\mathbf{x})\right\|$ corresponds to the magnitude of the parallel velocity component $\mathbf{u}_{\|}(\mathbf{x}) = \mathbf{u} - (\mathbf{u}\cdot\mathbf{n})\mathbf{n}$. Then the boundary conditions (18) and (19) can be rewritten as follows:

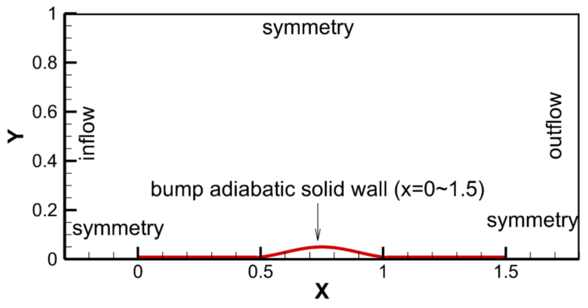
$$\mathbf{u}\cdot\mathbf{n}\big|_{\tilde{y}=0} = 0 \tag{26}$$

$$(\nu + \nu_{\mathrm{T}})\frac{\partial \mathbf{u}_{\|}}{\partial \mathbf{n}}\bigg|_{\tilde{y}=0} = \left(\frac{\mathbf{u}_{\|}}{u_{\|}}\bigg|_{\tilde{y}=0}\right) u_\tau^2(\tilde{x}, t). \tag{27}$$

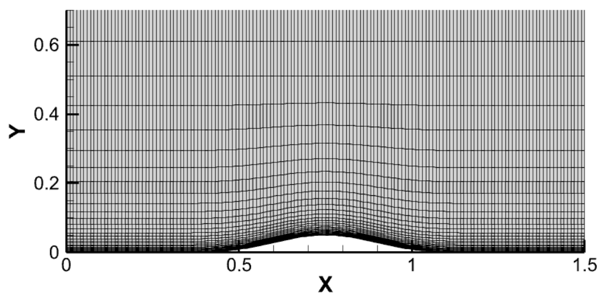
3. Numerical Method

The penalized wall function method, proposed in this paper, is implemented in the NOISEtte flow solver [17, 18]. The system of equations (1)–(3), (8) are discretized using vertex-centered finite-volume method, combined with quasi-one-dimensional variable reconstruction along a mesh edge (EBR-scheme) [3], used to increase the order of accuracy. The approximation of the viscous terms is based on Galerkin finite element method with linear basis functions. For time integration, an implicit three-layer 2nd order time integration method is used. At each stage of the time integration the spatially discretized system of nonlinear equations is solved using Newton's method with the linearized space-difference system of equations at each Newton's iteration solved using stabilized bi-conjugate gradient method (BI-CGSTAB) [34].

Each time integration step of the main system of equations (1)–(3), (8) is preceded by an implicit first-order time integration of Eq. (25), which for efficiency is discretized using first-order upwind-biased finite-difference method. The Gaussian function (23) is used for the localized



(a) Boundary conditions



(b) Computational mesh in the vicinity of the bump (mesh 2)

Figure 1. 2D Bump-in-channel flow problem

exchange layer masking function. The values of the normalized thickness of the exchange layer σ^+, the length scale l_s, the relaxation parameters η_f and η_s, and the initial value of u_τ used in the simulation are discussed in Section 4.

4. Simulations and Results

4.1. 2D Bump-in-channel Flow

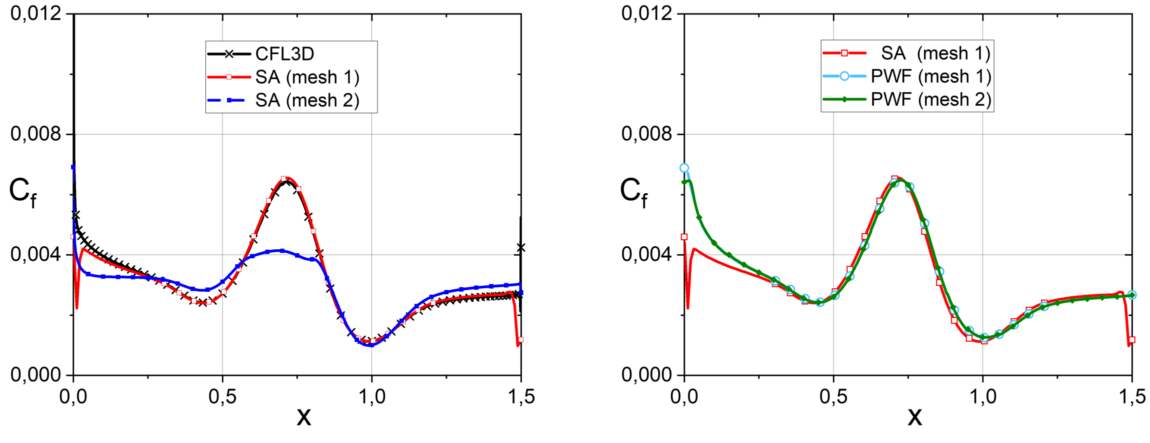
To demonstrate the effectiveness and accuracy of the proposed approach, the penalized wall function method is applied for the two dimensional simulation of compressible turbulent flow around a bump, also known as 2D bump-in-channel flow. The flow configuration is identical to the NASA turbulence model verification case [1]. The simulation results are compared with the reference solution [1], obtained using the CFL3D structured grid code [29] with the Spalart–Allmaras model [31].

The problem is non-dimensionalized identically to the reference case [1] using the following characteristic scales: the density of the undisturbed flow ρ_∞, the inflow velocity U_∞, the molecular viscosity at the inflow μ_∞, and the unit length L. The 2D bump-in-channel flow problem is solved for the Reynolds number $Re = 3 \times 10^6$ and the Mach number $M = 0.2$. A viscous compressible flow around an infinitely thin plate with an origin at the point $(0, 0)$ and a dimensionless length of 1.5 with the bump of the height of 0.05 is considered. The geometry of the bump is described as

$$y = \begin{cases} 0.05 \left\{\sin\left(\frac{\pi x}{0.9} - \frac{\pi}{3}\right)\right\}^4 & \text{if } 0.3 < x < 1.2, \\ 0 & \text{if } x <= 0.3 \text{ and } x >= 1.2. \end{cases} \tag{28}$$

The problem set up and the boundary conditions are shown in Fig. 1a. The boundary conditions are as follows. No-slip, adiabatic, zero eddy viscosity condition (12) are imposed at the solid plate surfaces for the RANS simulations using Spalart–Allmaras model [31]. For the penalized wall function method the no-slip and zero eddy viscosity conditions are replaced by no-penetration condition (18), wall shear stress condition (19), and condition (21) for the eddy viscosity.

The penalized wall function method is implemented as follows: the friction velocity field u_τ is defined in the entire computational domain with initial value of $u_\tau = 4 \times 10^{-2}$. The PWF equation (25) is integrated as a preprocessing step before each time integration step of the main



(a) RANS simulations with S-A model (b) PWF simulations

Figure 2. Distribution of the skin friction coefficient on the bump for two different resolutions

system of RANS equations (1)–(3), (8). The updated values of u_τ are subsequently used in the boundary condition (19) and the distance function definition (20) when solving the RANS equations for the next time step. The PWF equation (25) is solved for the localized exchange layer masking function χ_δ defined as the Gaussian function (23) with the normalized exchange layer thickness $\sigma^+ = 50$, the normalized distance at the exchange location is $\delta^+_{EL} = 100$, the nondimensional characteristic length scale $l_s = 1$ and time scales $\eta_s = 10^{-2}$ and $\eta_f = 10^{-2}$, and the Reichardt's law of the wall [28] for the wall function:

$$f_{\mathrm{Rei}}\left(y^+\right) = \frac{1}{\kappa}\ln\left(1+\kappa y^+\right) + 7.8\left[1 - \exp\left(-\frac{y^+}{11}\right) - \frac{y^+}{11}\exp\left(-\frac{y^+}{3}\right)\right], \tag{29}$$

where $\kappa = 0.41$ is the von Kármán constant.

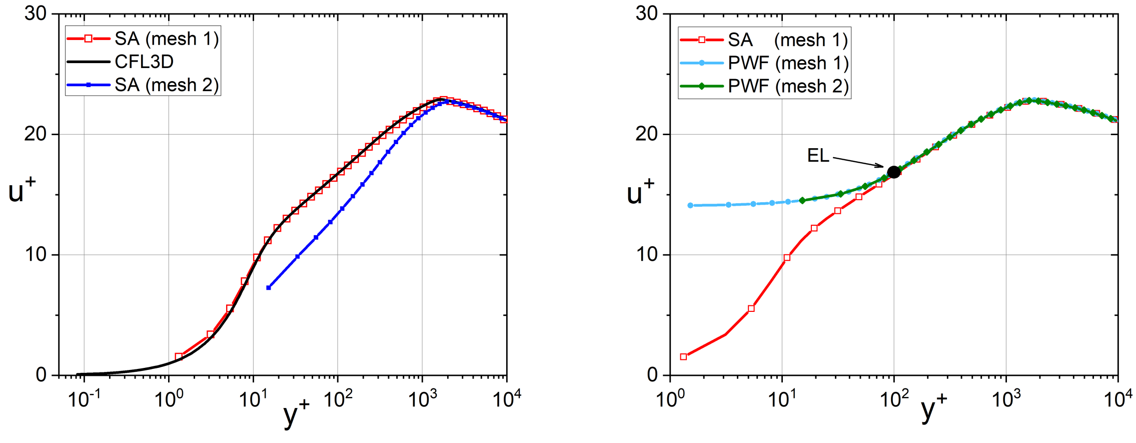
The simulations are carried out on structured meshes with the longitudinal grid size $\Delta h_{\|} \approx 10^{-2}$, which ensures the adequate resolution of the bump curvature. The wall normal mesh size is exponentially increasing with the growth factor of $q \approx 1.2$ and mesh resolution in the vicinity of the wall $\Delta h_\perp$.

To demonstrate the convergence of the PWF method, the simulations are carried out for two different near-wall resolutions: $\Delta h_\perp = 1 \times 10^{-5}$ and $\Delta h_\perp = 1 \times 10^{-4}$.

The mesh resolutions are chosen so that the RANS simulations with S-A model are well resolved for the first case, denoted as mesh 1, and unresolved for the second case, denoted as mesh 2. The zoomed-in view of the computational mesh in the vicinity of the bump, corresponding to the unresolved case, is shown in Fig. 1b, where for better visual perception every tenth vertical line is shown.

The results of the PWF simulations are compared with the reference solution [1] and with the results of the RANS simulations with the Spalart–Allmaras model [31], which are carried out on the same computational meshes using the same solver, but with no-slip and zero eddy viscosity (12) boundary conditions instead of the no-penetration condition (18), wall shear stress condition (19), and condition (21) used in the PWF method.

The effect of the wall model is demonstrated in Fig. 2, where skin friction coefficient C_f distribution on the bump for two different resolutions is shown for both the RANS and PWF simulations. As can be seen in Fig. 2a, the results of the resolved RANS simulations are in good



(a) RANS simulations with S-A model

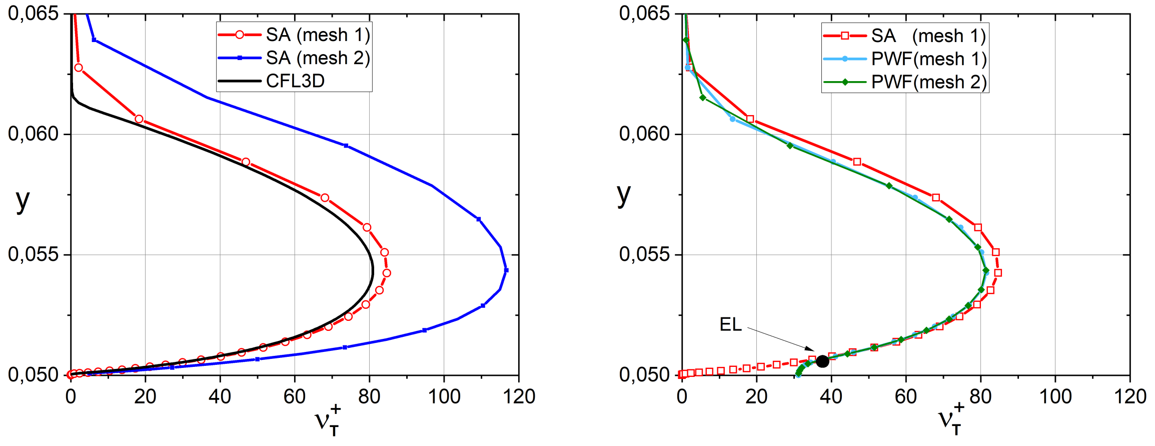
(b) PWF simulations

Figure 3. Streamwise velocity profiles at $x = 0.75$ for the 2D bump-in-channel flow problem for two different resolutions

agreement with the reference CFL3D solution [1], while for the second case the skin friction coefficient is significantly underestimated due to insufficient resolution of the boundary layer, required by the S-A model. In contrast, as can be seen in Fig. 2b, the skin friction coefficient distributions for the PWF simulations are practically identical for both resolutions and are in good agreement with the results of the resolved RANS simulations, which highlights the substantially lower near wall resolution requirements for the penalized wall function method compared to the resolved RANS simulations with S-A turbulence model. Note that the deviations of the skin friction coefficient at the leading and trailing edges of the plate are related to geometric singularities caused by the sudden application of either no-slip or wall shear stress conditions on the plate for the RANS and PWF simulations, respectively.

A similar trend is observed when considering velocity profiles. A comparison of the streamwise velocity profiles in the wall-normal direction at $x = 0.75$ is given in Figs. 3a and 3b, where the results of the RANS and PWF simulations, respectively, are shown. As can be seen in Fig. 3a the resolved RANS simulations with S-A model are in good agreement with the reference CFL3D solution [1], while for the unresolved case (mesh 2) the velocity profile is wrong. Substantially lower wall resolution requirements of the PWF method are demonstrated in Fig. 3b, which shows good agreement between the results of PWF simulations for all resolutions both in the boundary layer and in the outer region. Moreover, as can be seen in Fig. 3b, the PWF solution in the outer region, marked by the exchange location, is in excellent agreement with the results of the resolved RANS simulation with S-A turbulence model.

The distribution of relative turbulent viscosity $\nu_{\mathrm{T}}^{+} = \nu_{\mathrm{T}}/\nu$ in the wall-normal direction at $x = 0.75$ is shown in Figs. 4a and 4b for RANS and PWF simulations, respectively. As can be seen in Fig. 4a the value of turbulence viscosity is substantially higher for the unresolved case, compensating for the lack of wall normal resolution. The results of the resolved RANS simulations with the S-A turbulence model are slightly higher then the reference CFL3D solution [1], which is due to slightly lower resolution compared to the reference case. The eddy-viscosity for the PWF simulations for both resolutions are identical and in the outer region ($y^{+} > \delta_{EL}^{+}$) are slightly lower compared to the eddy-viscosity of the resolved RANS simulation with the S-A turbulence



(a) RANS simulations with S-A model (b) PWF simulations

Figure 4. Eddy-viscosity profiles at $x = 0.75$ for the 2D bump-in-channel flow problem for two different resolutions

model. This discrepancy is related to limiting the wall distance according to Eq. (20), which, in turn, results in almost constant eddy-viscosity in the inner region ($y^+ < \delta^+_{EL}$).

Conclusions

A novel penalized wall function method is proposed for simulations of wall-bounded compressible turbulent flows. The new approach, similar to classical wall function methods, is based on the Reynolds-averaged Navier–Stokes (RANS) equations to model the outer region of the turbulent boundary layer, while approximating the inner part by an analytic wall function. The differential formulation to match the outer and wall function solutions is reformulated in a form of a generalized characteristic-based volume penalization method for the friction velocity to model the transfer of the shear stress from the outer region of the boundary layer to the wall and to impose the wall-stress boundary conditions on the RANS solution. The exchange location in the new formulation is specified implicitly through a localized source term in the boundary layer equation, which eliminates the need to interpolate the solution to the exchange location as well as the need to explicitly determine the position of the exchange location. The wall-stress condition is determined by solving an auxiliary equation for the friction velocity, ensuring the correct matching of the RANS and the wall function solutions at the exchange layer. The penalized wall function method is demonstrated for Reynolds-averaged Navier–Stokes equations with the Spalart–Allmaras turbulence model, but can be used in conjunction with any turbulence model that allows transfer of shear stress from the outer region of the boundary layer to the wall.

The proposed method noticeably reduces the near-wall mesh resolution requirements without significant modification of the RANS solver. The formulation of the penalized wall function method is general and can be used in context of any numerical method based on either structured or unstructured meshes.

The effectiveness of the developed penalized wall function method is demonstrated for two-dimensional bump-in-channel flow, which is characterized by the presence of a significant longitudinal pressure gradient without flow separation. The simulations demonstrate the sufficient accuracy of the PWF solution on grids with 10 times coarser near wall resolution compared to the resolution required by the the Spalart–Alamaras model with no-slip boundary conditions.

Further development of PWF method includes its generalization to problems with strong pressure gradients and flow separation, which would require the reformulation of the method in terms of differential equilibrium and non-equilibrium wall functions.

Acknowledgements

This work has been carried out using computing resources of the federal collective usage center Complex for Simulation and Data Processing for Mega-science Facilities at NRC "Kurchatov Institute", http://ckp.nrcki.ru/.

References

1. NASA Langley research center turbulence modeling resource, `https://turbmodels.larc.nasa.gov`, accessed: 2022-11-07

2. Abalakin, I.V., Vasilyev, O.V., Zhdanova, N.S., Kozubskaya, T.K.: Characteristic based volume penalization method for numerical simulation of compressible flows on unstructured meshes. Comput. Math. and Math. Phys. 61(8), 1315–1329 (2021). `https://doi.org/10.1134/S0965542521080029`

3. Bakhvalov, P., Abalakin, I., Kozubskaya, T.: Edge-based reconstruction schemes for unstructured tetrahedral meshes. Int. J. Numer. Meth. Fluids 81(6), 331–356 (2016). `https://doi.org/10.1002/fld.4187`

4. Bardina, J., Huang, P., Coakley, T., *et al.*: Turbulence modeling validation. In: 28th Fluid dynamics conference. p. 2121 (1997)

5. Beaugendre, H., Morency, F.: Penalization of the Spalart–Allmaras turbulence model without and with a wall function: Methodology for a vortex in cell scheme. Computers & Fluids 170, 313–323 (Jul 2018), `https://hal.inria.fr/hal-01963687`

6. Beaugendre, H., Morency, F.: Penalization of the Spalart–Allmaras turbulence model without and with a wall function: Methodology for a vortex in cell scheme. Computers & Fluids 170, 313–323 (2018). `https://doi.org/10.1016/j.compfluid.2018.05.012`

7. Bodart, J., Larsson, J.: Wall-modeled large eddy simulation in complex geometries with application to high-lift devices. Annual Research Briefs, Center for Turbulence Research, Stanford University pp. 37–48 (2011)

8. Brown-Dymkoski, E., Kasimov, N., Vasilyev, O.V.: A characteristic based volume penalization method for general evolution problems applied to compressible viscous flows. J. Comp. Phys. 262, 344–357 (2014). `https://doi.org/10.1063/1.4825260`

9. Brown-Dymkoski, E., Kasimov, N., Vasilyev, O.V.: A characteristic-based volume penalization method for arbitrary mach flows around solid obstacles. In: Fröhlich, J., Kuerten, H., Geurts, B.J., Armenio, V. (eds.) Direct and Large-Eddy Simulation IX. pp. 109–115. Springer, Cham (2015). `https://doi.org/10.1007/978-3-319-14448-1_15`

10. Cai, S.G., Degrigny, J., Boussuge, J.F., Sagaut, P.: Coupling of turbulence wall models and immersed boundaries on cartesian grids. J. Comp. Phys. 429, 109995 (2021). `https://doi.org/10.1016/j.jcp.2020.109995`

11. Craft, T., Gant, S., Gerasimov, A., *et al.*: Development and application of wall-function treatments for turbulent forced and mixed convection flows. Fluid Dyn. Res. 38(2), 127–144 (2006). https://doi.org/10.1016/j.fluiddyn.2004.11.002

12. Dhamankar, N., Blaisdell, G., Lyrintzis, A.: Implementation of a wall-modeled sharp immersed boundary method in a high-order large eddy simulation tool for jet aeroacoustics. In: 54th AIAA Aerospace Sciences Meeting (01 2016). https://doi.org/10.2514/6.2016-0257

13. Duben, A.P., Abalakin, I.V., Tsvetkova, V.O.: On boundary conditions on solid walls in viscous flow problems. Math. Models and Comput. Simul. 13(4), 591–603 (2021). https://doi.org/10.1134/S2070048221040128

14. Durbin, P.A., Reif, B.A.P.: Statistical Theory and Modeling for Turbulent Flows. Wiley (2001)

15. Froehlich, J., von Terzi, D.: Hybrid LES/RANS methods for the simulation of turbulent flows. Progress in Aerospace Sciences 44(5), 349–377 (2008). https://doi.org/10.1016/j.paerosci.2008.05.001

16. Gatski, T.B., Hussaini, M.Y., Lumley, J.L.: Simulation and Modeling of Turbulent Flows. Oxford (1996)

17. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Comput. Phys. Commun. 271, 108231 (2022). https://doi.org/10.1016/j.cpc.2021.108231

18. Gorobets, A., Duben, A.: Technology for supercomputer simulation of turbulent flows in the good new days of exascale computing. Supercomput. Front. Innov. 8(4), 4–10 (Feb 2021). https://doi.org/10.14529/jsfi210401

19. Kasimov, N., Dymkoski, E., De Stefano, G., Vasilyev, O.V.: Galilean-invariant characteristic-based volume penalization method for supersonic flows with moving boundaries. Fluids 6(8) (2021). https://doi.org/10.3390/fluids6080293

20. Kawai, S., Larsson, J.: Wall-modeling in large eddy simulation: length scales, grid resolution, and accuracy. Phys. Fluids. 24(1), 015105 (2012). https://doi.org/10.1063/1.3678331

21. Kawai, S., Larsson, J.: Dynamic non-equilibrium wall-modeling for large eddy simulation at high Reynolds numbers. Phys. Fluids. 25(1), 015105 (2013). https://doi.org/10.1063/1.4775363

22. Liu, Q., Vasilyev, O.V.: Hybrid adaptive wavelet collocation – Brinkman penalization method for unsteady RANS simulations of compressible flow around bluff bodies. In: 36th AIAA Fluid Dynamics Conference and Exhibit, San Francisco, California, USA, June 5–8, 2006 (2006). https://doi.org/10.2514/6.2006-3206

23. Liu, Q., Vasilyev, O.V.: A Brinkman penalization method for compressible flows in complex geometries. J. Comp. Phys. 227(2), 946–966 (2007). https://doi.org/10.1016/j.jcp.2007.07.037

24. Moin, P., Mahesh, K.: Direct numerical simulation: A tool in turbulence research. Annual Rev. Fluid Mech. 30, 539–578 (1998). https://doi.org/10.1146/annurev.fluid.30.1.539

25. Nichols, R.H., Nelson, C.C.: Wall function boundary conditions including heat transfer and compressibility. AIAA Journal 42(6), 1107–1114 (2004). https://doi.org/10.2514/1.3539

26. Park, G.I., Moin, P.: An improved dynamic non-equilibrium wall-model for large eddy simulation. Phys. Fluids. 26(1), 37–48 (2014). https://doi.org/10.1063/1.4861069

27. Patankar, S.V., Spalding, D.B.: Heat and Mass Transfer in Boundary Layers. Morgan-Grampia (1968)

28. Reichardt, H.: Vollständige darstellung der turbulenten geschwindigkeitsverteilung in glatten leitungend. Zeitschrift für Angewandte Mathematik und Mechanik 31(7), 208–219 (1951)

29. Rumsey, C., Gatski, T., Sellers, W., *et al.*: Summary of the 2004 CFD validation workshop on synthetic jets and turbulent separation control. In: 2nd AIAA Flow Control Conference, Portland, Oregon, June 28 – July 1, 2004. p. 2217 (2004). https://doi.org/10.2514/6.2004-2217

30. Shur, M.L., Spalart, P.R., Strelets, M.K., Travin, A.K.: A hybrid RANS-LES approach with delayed-DES and wall-modelled LES capabilities. Int. J. Heat Fluid Flow 29(6), 1638–1649 (2008). https://doi.org/10.1016/j.ijheatfluidflow.2008.07.001

31. Spalart, P.R., Allmaras, S.R.: A one equation turbulence model for aerodinamic flows. AIAA journal 94 (1992). https://doi.org/10.2514/6.1992-439

32. Spalart, P.R., Deck, S., Shur, M.L., *et al.*: A new version of detached-eddy simulation, resistant to ambiguous grid densities. Theoretical and computational fluid dynamics 20(3), 181–195 (2006). https://doi.org/10.1007/s00162-006-0015-0

33. Spalart, P.R., Jou, W.H., Strelets, M., *et al.*: Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach. In: First AFOSR international conference on DNS/LES, Ruston, Louisiana. vol. 1, pp. 4–8. Greyden Press, Columbus, OH (1997)

34. van der Vorst, H.A.: BI-CGSTAB: A fast and smoothly converging variant of BI-CG for the solution of nonsymmetric linear systems. SIAM J. Sci. Stat. Comput. 13(2), 631–644 (1992). https://doi.org/10.1137/0913035

35. Wilcox, D.C.: Formulation of the $k - \omega$ turbulence model revisited. AIAA Journal 46(11), 2823–2838 (2008). https://doi.org/10.2514/1.36541

36. Xiao, H., Jenny, P.: A consistent dual-mesh framework for hybrid LES/RANS modeling. J. Comp. Phys. 231(4), 1848–1865 (FEB 20 2012). https://doi.org/10.1016/j.jcp.2011.11.009

37. Zhdanova, N.S., Abalakin, I.V., Vasilyev, O.V.: Generalized Brinkman volume penalization method for compressible flows around moving obstacles. Math. Models. and Comput. Simul. 14(5), 716–726 (2022). https://doi.org/10.1134/S2070048222050180

DOI: 10.14529/jsfi220407

Hybrid Dynamic Mesh Redistribution – Immersed Boundary Method for Acoustic Simulation of Flow Around a Propeller*

Vladimir G. Bobkov[1], *Tatiana K. Kozubskaya*[1], *Liudmila N. Kudryavtseva*[1,2], *Valeriia O. Tsvetkova*[1]

A novel hybrid dynamic mesh redistribution – immersed boundary method for simulation of turbulent flows around rotating obstacles of complex geometry and analysis of tonal acoustics is proposed. The feasibility of the approach is demonstrated by considering a drone propeller problem. The results of three-dimensional Reynolds-averaged Navier–Stokes simulations using the proposed approach are compared to the results of body-fitted unstructured simulations in non-inertial reference frame. The dynamic mesh redistribution method allows the reposition of mesh points taking into account the shape of the moving body while retaining the mesh topology. The cell size and quality of the dynamically redistributed mesh strongly depend on the curvature of the body surface. The position and shape of the moving obstacle is prescribed by a distance function defined on an adaptive octree. The results of simulations using the proposed method are in good agreement with both the results of body-fitted simulations and the experimental data.

Keywords: moving adaptive mesh, immersed boundary method, drone rotor, rotor acoustics, unstructured mesh, turbulent flow.

Introduction

The problem of flow simulation around moving obstacles or body parts is of high importance in computational aerodynamics and aeroacoustics. There is a number of different modeling techniques and mesh changing methods that are currently being developed to approach the above-described problem. Most popular approaches preserve the classical body-fitted (BF) meshes, requiring boundary nodes to coincide with the obstacle surface. For instance, mesh deformation method, based on quasi one-dimensional elastic media [6], is the least computationally expensive body-fitted approach, which allows efficient deformation of the computational mesh governed by the body motion. The main disadvantage of the mesh deformation method is its inability to handle large obstacle displacements leading to mesh deterioration and/or entanglement. An alternative approaches based on mesh recomputation [7] or adaptive mesh refinement [8], regardless of their computational efficiency, introduce interpolation errors and significantly complicate the process of parallelization, since the topology and mesh size can change over time. The Chimera methods [9] utilizing two or more superimposed computational meshes also require interpolation and data exchange between the meshes.

This paper describes a novel hybrid dynamic mesh redistribution – immersed boundary method (DMR-IBM), capable of simulating compressible flows around arbitrary number of obstacles, either moving according to their own laws or displaced under the action of aerodynamic forces. The feasibility of combining the immersed boundary method with the dynamic mesh redistribution is evaluated for the acoustic simulations of a compressible flow around an isolated drone rotor. The immersed boundary approach is based on the Brinkman penalization method [1–5], where the obstacle is modeled by introducing additional source terms into the

*One of the selected papers of The Ninth Russian Conference "Computational Experiment in Aeroacoustics and Aerodynamics" (CEAA2022)

[1]Keldysh Institute of Applied Mathematics, Moscow, Russian Federation

[2]Dorodnicyn Computing Center FRC CSC RAS, Moscow, Russian Federation

governing equations describing the evolution of compressible viscous flow. These sources define the body as a continuous porous medium with low permeability. The source terms are non-zero inside the moving obstacle and zero – outside. Thus, the use of the immersed boundary method makes it possible to define the external flow problem in a simply connected domain, which opens up the possibility of using the dynamic mesh redistribution method to position mesh points in the vicinity of the surface of the moving obstacle, while maintaining the mesh topology. In this paper, we consider the problem of aeroacoustics simulation of a drone propeller taken as an example of a rotating body of a very complex geometry. The hybrid DMR-IBM was previously formulated for two-dimensional flows and tested for the rotating projection of the propeller [5]. In this paper, the DMR-IBM is generalized to three dimensions and is verified for the problem of tonal acoustics of a propeller by comparing the results of the simulations against the data obtained using the classical body-fitted approach in non-inertial frame of reference. The DMR-IBM implemented for a propeller located inside of a puck-shaped domain with the mesh points dynamically redistributed to follow the motion of the propeller and to resolve the boundary layer around it. Such an approach would allow, in future, to use DMR-IBM for simulation of flows around multiple rotating objects located within its own puck-shaped subdomian belonging to a larger computational domain, while approximating fuselage using a body-fitted mesh approach. An example of such a problem would be a flow around vehicle with several rotors, e.g., a quad-copter.

A complex geometry of the propeller is characterized by sharp corners, very thin parts of blades, and high curvature regions. To provide a sufficiently high resolution mesh near the rotating propeller surface by means of dynamic mesh redistribution method is quite challenging problem. To overcome this challenge, a number of novel methods has been developed. To start, the initial mesh with high node density in the region of the propeller rotation is constructed. Then the control mesh adaptation metric is developed. This metric depends on the distance function and its gradient as well as on parameters of the immersed body (i.e. propeller) and its surface, such as curvature and distances to the internal and external medial axes.

The article is organized as follows. Section 1 gives the information on the problem formulation. In Section 2 we discuss the choice of the mathematical model for two approaches. Section 3 contains information about the computational set-up, including details of the numerical method, computational meshes, mesh adaptation algorithm and its parallel implementation in corresponding subsections. Section 4 is devoted to the numerical results of aerodynamics and acoustics. Conclusion summarizes the study and points directions for further work.

1. Problem Formulation

The problem considered in this paper is similar to the APC Slow Flyer 10x4.7 problem, namely, the small-scaled UAV rotor problem studied in [10]. Similar to Ref. [11], the drone problem with the rotor and hub radii of $R = 0.127\ m$ and $r = 0.0127\ m$, respectively, is studied. The blade of the rotor is based on the Eppler E63 airfoil (blade inner part) and Clark-Y airfoil (near blade tip) with non-linear twist and chord spanwise distribution (Fig. 1). The propeller is designed for 2 000–20 000 rotations per minute (RPM), while the regime under consideration is 4 000 RPM, which corresponds to the tip velocity $V_{tip} = 53.2\ m/s$. In the computational setup, the size of the propeller is normalized by the maximum length of the chord of the blade $b = 0.0287\ m$. The Reynolds number, defined using the blade tip velocity and the blade maximum chord, is $Re = \rho_0 V_{tip} b/\mu_0 \approx 10^5$, where $\rho_0 = 1.204\ kg/m^3$ and $\mu_0 = 1.815 \times 10^{-5}\ Pa \cdot s$

corresponding to the parameters of the air at temperature of $T_0 = 293.15K$. The tip Mach number is $M = V_{tip}/\sqrt{\frac{\gamma \mathcal{R} T_0}{\mathcal{M}}} = 0.156$, where γ is the adiabatic index, $\mathcal{R}$ is the gas constant and $\mathcal{M}$ is the molar mass of the gas.

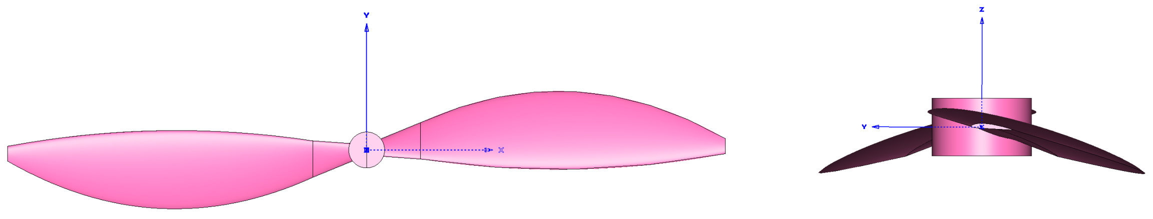


Figure 1. The rotor geometry

2. Mathematical Model

Large Reynolds number, high velocity flows are typical for aviation applications. Direct Numerical Simulation (DNS) of such flows requires high resolution computations both in space and time that are beyond the current realm despite the continuing growth of the performance of supercomputers. There is a number of approaches which can be applied to model turbulent flow near propeller, including RANS, LES and hybrid RANS-LES approaches. It should be mentioned that there is no full understanding yet that correct modeling of aerodynamics using presented technique is feasible. We are aiming to simulate rotor acoustics. So, differences that [12] presents by comparing RANS and RANS-LES for simulation of flow around rotating propeller do not make big differences in terms of our aims. For that reason, the turbulent flow simulation uses compressible Reynolds-averaged Navier–Stokes (RANS) equations with the Spalart–Allmaras turbulence model [13]. The system of RANS equations is written in the form of conservation laws for the vector $\mathbf{Q}$ of conserved variables

$$\mathbf{Q} = (\rho, \rho\mathbf{u}, E, \rho\tilde{\nu})^T,$$

where $\mathbf{u} = (u_1, u_2, u_3)$ is the velocity vector, ρ is the density, $E = \rho\mathbf{u}^2/2 + \rho\varepsilon$ is the total energy, ε is the specific internal energy, p is the pressure defined by ideal gas equation $p = \rho\varepsilon(\gamma - 1)$, $\gamma = 1.4$ is the adiabatic exponent, $\tilde{\nu}$ is the evolutionary variable which is used to determine the turbulent viscosity μ_T according to the Spalart–Allmaras model:

$$\mu_T = \rho\tilde{\nu}\frac{\chi^3}{\chi^3 + 357.911}, \quad \chi = \frac{\rho\tilde{\nu}}{\mu},$$

where μ is the coefficient of dynamic molecular viscosity.

The system of Reynolds-averaged Navier–Stokes equations can be written in the following vector form:

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot (\mathcal{F}^C(\mathbf{Q}) - \mathcal{F}^D(\mathbf{Q}, \nabla\mathbf{Q})) = \mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q}). \quad (1)$$

System (1) includes composite vectors $\mathcal{F}^C$ and $\mathcal{F}^D$, each component of which $\mathbf{F}_i^C$ and $\mathbf{F}_i^D$ in coordinate direction x_i $(i = 1, 2, 3)$ represents the convective transport and diffusion flux vectors, respectively. Operator $(\nabla\cdot)$ is the divergence operator.

The convective transport flux vector is given as a function of the physical variables ρ, $\mathbf{u}$, p

$$\mathbf{F}_i^C(\mathbf{Q}) = (\rho u_i, \rho u_i \mathbf{u} + p\mathbf{I}, (E+p)u_i, \rho\tilde{\nu}u_i)^T, \tag{2}$$

where $\mathbf{I}$ is the identity matrix. The diffusion flux vector is defined as a function of physical variables and their gradients as

$$\mathbf{F}_i^D(\mathbf{Q}, \nabla\mathbf{Q}) = \left(0,\ \tau_{i1},\ \tau_{i2},\ \tau_{i3},\ \tau_{ij}u_j + q_i,\ \frac{3}{2}(\mu + \rho\tilde{\nu})\frac{\partial\tilde{\nu}}{\partial x_i}\right)^T, \tag{3}$$

where the components of the viscous tensor of viscous stresses and the heat flux vector can be written as follows:

$$\tau_{ij} = (\mu + \mu_T)\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i} - \frac{2}{3}\delta_{ij}\frac{\partial u_i}{\partial x_i}\right),\ q_i = \left(\frac{\mu}{\mathrm{Pr}} + \frac{\mu_T}{\mathrm{Pr}_T}\right)\frac{\partial\varepsilon}{\partial x}, \tag{4}$$

where δ_{ij} is the Kronecker symbol, μ is the molecular viscosity coefficient, $\mathrm{Pr} = 0.72$ and $\mathrm{Pr}_T = 1$ are the molecular and turbulent Prandtl numbers, respectively.

Vector $\mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q})$ is a source term describing the influence of the external forces that are not related to the transfer processes of the target variables $\mathbf{Q}$:

$$\mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q}) = (0,\ 0,\ 0,\ P_\nu(\mathbf{Q}, \nabla\mathbf{Q}) - Y_\nu(\mathbf{Q}, \nabla\mathbf{Q}) + 0.992\nabla\tilde{\nu}\cdot\nabla\tilde{\nu})^T. \tag{5}$$

The detailed definition of terms $P_\nu(\mathbf{Q}, \nabla\mathbf{Q})$, $Y_\nu(\mathbf{Q}, \nabla\mathbf{Q})$ describing respectively the generation and dispersion of turbulence is given in paper [13].

Let us formulate the immersed boundary condition for system (1). At the boundary between a solid Ω_B and a medium Ω_f there is a no-slip condition:

$$u|_{\partial\Omega_B} = V. \tag{6}$$

Condition (6) is defined by Brinkman penalization method [1]. The Brinkman penalization modifies the right-hand side of the system (1) by adding the extra source terms so that the new vector of source term becomes as:

$$\mathbf{S}^{penal}(\mathbf{Q}, \nabla\mathbf{Q}) = \mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q}) + \left(0,\ \frac{\chi}{\eta}\rho(u_i - u_{Bi}),\ \frac{\chi}{\eta}\rho u_i(u_i - u_{Bi}),\ 0\right), \tag{7}$$

where χ defines the body location at every time moment as follows:

$$\chi(t) = \begin{cases} 1,\ x \in \overline{\Omega}_B(t) \\ 0,\ x \in \Omega_f(t). \end{cases} \tag{8}$$

Parameter η determines the rate of the relaxation of the flow velocity to the velocity of the moving body. In this paper the penalization parameter is equal to 10^{-4}. When we use the standard body-fitted approach to simulate the flow over the rotating propeller we solve the RANS equations in non-inertial frame of reference [14]. Let $\mathbf{V} = (V_1, V_2, V_3)^T = (\boldsymbol{\omega} \times \mathbf{r})$ be the peripheral propeller speed determined by angular velocity vector $\boldsymbol{\omega}$ and radius vector of the point in medium. Following this notation, the system of RANS equations (1) can be rewritten as

$$\frac{\partial\mathbf{Q}}{\partial t} + \nabla\cdot(\mathcal{F}^C(\mathbf{Q}) - \mathcal{F}^R(\mathbf{Q}) - \mathcal{F}^D(\mathbf{Q}, \nabla\mathbf{Q})) = \tilde{\mathbf{S}}(\mathbf{Q}, \nabla\mathbf{Q}), \tag{9}$$

where $\mathcal{F}^R$ is the rotation transport flux with the components

$$\mathbf{F}_i^R(\mathbf{Q}) = (\rho V_i,\ \rho u_i \mathbf{V},\ E V_i,\ \rho\tilde{\nu} V_i)^T. \tag{10}$$

Source term (5) is now dependent on angular velocity:

$$\tilde{\mathbf{S}}\left(\mathbf{Q}, \nabla\mathbf{Q}\right) = (0,\ \rho\,\boldsymbol{\omega}\times\mathbf{u},\ 0,\ P_\nu\left(\mathbf{Q},\nabla\mathbf{Q}\right) - Y_\nu\left(\mathbf{Q},\nabla\mathbf{Q}\right) + 0.992\nabla\tilde{\nu}\cdot\nabla\tilde{\nu})^T. \tag{11}$$

From the point of view of an observer in the inertial coordinate system, the system of equations (9)–(11) with flux vectors (2) and (3) describes the change in conservative variables due to their convective and diffusive transport in a medium rotating at speed $\mathbf{V}$, the influence of the pressure gradient and the rotation of the velocity vector at the azimuthal angle $\psi(t) = -\left|\boldsymbol{\omega}\right| t$.

3. Computational Set-up

3.1. Numerical Method

The system of RANS equations (1) is solved using the fifth order Edge-based Reconstruction (EBR5) scheme [15–17]. The higher accuracy is achieved through the quasi-one-dimensional reconstructions of variables on the extended stencils oriented along the mesh edges. The approximation is built in such a way that being applied to translationally invariant meshes (i.e., meshes that transform into themselves when translated by an edge vector) the EBR5 scheme provides the fifth order of accuracy.

The EBR scheme for the Euler equations presents a vertex-centered method with physical and conservative variables defined at the mesh vertices around which the computational cells of the dual mesh are built. Thus, a finite-volume approximation of convective flows is constructed for dual mesh cells acting as finite volumes. The viscous terms of the RANS equations are approximated using the finite-element method with the linear basis functions.

For the time integration, the implicit second-order scheme with Newton iterations is used to solve the nonlinear system of algebraic equations resulting from the space discretization. At each Newtonian iteration, the corresponding system of linear equations is solved using the stabilized biconjugate gradient method.

3.2. Dynamic Mesh Redistribution Method

To build a dynamic mesh redistribution method, we consider a computational mesh as an elastic material which is deformed when zones of mesh compression follow the boundaries of moving obstacles. The time-dependent elastic deformation is defined by a special mapping $\mathbf{x}(\boldsymbol{\xi}, t) : R^d \times R \to R^d$. Let C define the Jacobian matrix of mapping $\mathbf{x}(\boldsymbol{\xi}, .)$, where $c_{ij} = \dfrac{\partial x_i}{\partial \xi_j}$. Let $\mathbf{x}^n(\boldsymbol{\xi})$ denote the mapping of the initial mesh onto the mesh at time t^n.

We consider the mapping of a regular tetrahedron T_i with the vertices coordinates $\mathbf{h_0}$, $\mathbf{h_1}$, $\mathbf{h_2}$, $\mathbf{h_3}$ in Lagrangian domain $\Omega_{\boldsymbol{\xi}}$ to a tetrahedron with Eulerian coordinates of the vertices $\mathbf{p_0}$, $\mathbf{p_1}$, $\mathbf{p_2}$, $\mathbf{p_3}$ in domain $\Omega_{\mathbf{x}}$. The Eulerian coordinates are the desired coordinates in the computational domain. For our problem formulation $\Omega_{\boldsymbol{\xi}}$ is some speculative domain that consists of regular tetrahedrons and $vol(\Omega_{\boldsymbol{\xi}}) = vol(\Omega_{\mathbf{x}})$. At the time moment t for each vertex of the tetrahedron Jacobian matrix of this mapping can be written as $\nabla_{\boldsymbol{\xi}}\mathbf{x}(\boldsymbol{\xi}, t)\mathbf{H}^{-1}$, where $\nabla_{\boldsymbol{\xi}}\mathbf{x}(\boldsymbol{\xi}, t) = (\mathbf{p_1} - \mathbf{p_0},\ \mathbf{p_2} - \mathbf{p_0},\ \mathbf{p_3} - \mathbf{p_0})$ and $\mathbf{H} = (\mathbf{h_1} - \mathbf{h_0},\ \mathbf{h_2} - \mathbf{h_0},\ \mathbf{h_3} - \mathbf{h_0})$. Here $\boldsymbol{\xi}$ are local barycentric coordinates inside parametric tetrahedron.

To keep track of moving body we consider metric in Eulerian coordinates which depends on body geometry and time. Let $\mathbf{G}(\mathbf{x}, t)$ denote metric tensor and $\mathbf{Q} = \mathbf{Q}(\mathbf{x}, t)$ be an arbitrary matrix factorization of metric tensor $\mathbf{G}(\mathbf{x}, t)$, defined by

$$\mathbf{Q}^T\mathbf{Q} = \mathbf{G}(\mathbf{x}, t), \ \det \mathbf{Q} > 0.$$

To introduce the variational problem for mesh optimization one can assume that $\mathbf{x}^n(\boldsymbol{\xi})$ is a quasi-isometric piecewise affine one-to-one mapping. We find the mapping of the initial mesh onto the mesh at time $t = t^{n+1}$ as minimizer of this semi-discrete functional

$$F(\mathbf{x}(\boldsymbol{\xi}, t), \mathbf{x}^n(\boldsymbol{\xi})) = \sum\nolimits_i \int\limits_{T_i} W(\mathbf{Q}(\mathbf{x}^n(\boldsymbol{\xi}), t)\nabla_{\boldsymbol{\xi}}\mathbf{x}(\boldsymbol{\xi}, t)\mathbf{H}^{-1}) \det \mathbf{H} d\boldsymbol{\xi}, \tag{12}$$

where $\mathbf{C} = \mathbf{Q}\nabla_{\boldsymbol{\xi}}\mathbf{x}(\boldsymbol{\xi}, t)\mathbf{H}^{-1}$ is the Jacobian matrix of mapping from regular tetrahedron T_i to i-th tetrahedron in metric space.

In functional (12) function $W(\mathbf{C})$ defines polyconvex elastic potential (internal energy), which is a weighted sum of the shape distortion measure and the volume distortion measure:

$$W(\mathbf{C}) = (1-\theta)\frac{\frac{1}{d}\operatorname{tr}(\mathbf{C}^T\mathbf{C})}{\det \mathbf{C}^{2/d}} + \frac{1}{2}\theta\left(\frac{1}{\det \mathbf{C}} + \det \mathbf{C}\right). \tag{13}$$

In most cases we set $\theta = 4/5$. The elastic potential is minimum when the deformation is isometric, meaning that only rotation and translation are allowed. The details of the current variational approach are written in [18].

To use the mesh vertices effectively, the adaptation should be anisotropic. At the same time, it is necessary to qualitatively capture the surface of the body and its features. The correct definition of the adaptation metric requires normal and tangential directions at each point $\mathbf{p}$ which are defined by the isosurface of signed distance function $u(\mathbf{x}, t)$ passing through $\mathbf{p}$ and stretching coefficients σ_i along these directions. When σ_i is large, the local cell size in correspondent direction is small. Function $\sigma_1 = \sigma_{normal}(\mathbf{x}, t) = \phi(u(\mathbf{x}, t))$ defines the mesh stretching in the normal direction to the body and $\sigma_2 = \sigma_{2,tangential}(\mathbf{x}, t)$ and $\sigma_3 = \sigma_{3,tangential}(\mathbf{x}, t)$ define the spatial distribution of the anisotropy. We require the highest anisotropy in a thin boundary layer near the body, then the anisotropy is partially reduced to zero away from the body. Here 1D function $\phi(\cdot) : R^1 \to R^1$ is a hyperbola, which defines the mesh density in normal direction and guarantees transition from small mesh cells to large mesh cells with the prescribed rate of growth of mesh cell size. The definitions of σ_2 and σ_3 require extra information about the body shape such as principal curvatures, principal normal directions, and distances to the medial axis. All this volumetric data is gained during the preprocessing stage. Parameters σ_2 and σ_3 are used for defining of the control metric, so that mesh cells are close to isotropic near the sharp edges and anisotropic in the tangential surface near the regions close to a plane. The complete procedure of geometry preparation and the algorithm for constructing the anisotropic mesh adaptation metric is described in detail in [19].

We solve our adaptation mesh problem in a small cylinder that encompasses propeller geometry. Vertices on the boundary of this small cylinder are fixed. To achieve a good mesh refinement at all the edges of the propeller geometry and provide continuity of point distribution through the fixed boundary, the initial mesh is specially deformed. The vertices in the small cylinder are slightly redistributed to follow up the features of the body shape: an area near the hub and the region where the tip of the blade moves have higher mesh density as shown in Fig. 2. To account for that nonuniformity, the background metric is introduced. The background metric is a spherical metric where the eigenvalues are defined as cubic root of a ratio of average adjacent cell volume in parametric space to the correspondent volume in physical space. So now there are two metrics: background metric and adaptation metric $\mathbf{G_x}$. For the metric interpolation the log-Euclidian framework [20] is used. The paper [20] proposes introducing the metric logarithm and the commutative logarithm addition.

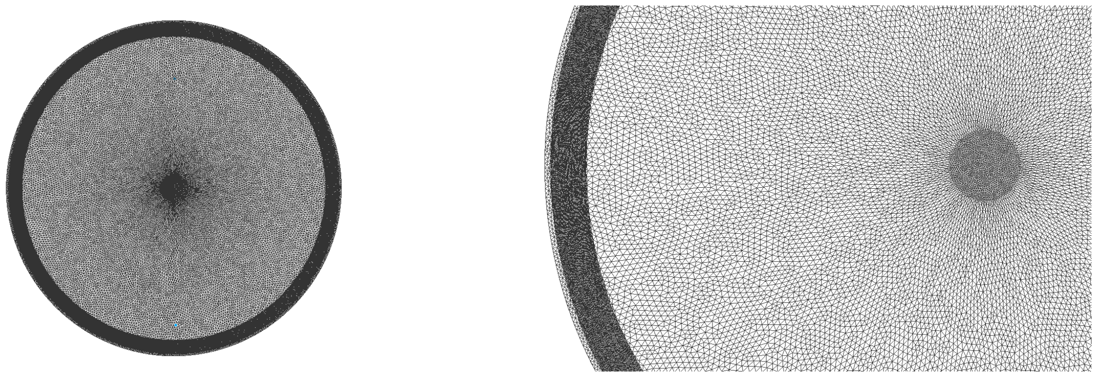


Figure 2. The initial mesh in a puck-shaped domain to start the adaptation to the propeller surface

Metric tensor $\mathbf{G_x}$ is evaluated in the mesh vertices which is crucial to the stability of very thin and highly compressed mesh layers. To solve the optimization problem on the each time step, we apply the preconditioned gradient descent technique [21] where the minimization direction is computed via an approximate solution of the linear system with the reduced Hessian matrix of the functional. The final mesh increment along the minimization direction is computed via the 1D search technique. The full description of the variational approach of the adaptation algorithm is presented in [18].

3.3. Computational Meshes

3.3.1. Mesh for body-fitted approach

The computational mesh for the body-fitted approach is a cylinder with radius $10R$ and height $30R$ (Fig. 3). R is the propeller radius. Propeller was placed in the center of the computational domain. The mesh near the rotor surface is filled with prismatic elements to resolve the boundary layer. A height of the near-surface element is chosen to meet $y^+ < 1$ criteria in the CFD simulation, meaning the viscous sublayer is resolved. y^+ is the dimensionless wall distance defined as $y^+ = \dfrac{yu_\tau}{\nu}$, where $u_\tau = \sqrt{\dfrac{\tau_\omega}{\rho}}$ is friction velocity, τ_ω is the wall shear stress, y is the absolute distance and ν is the kinematic viscosity.

The rest space between prismatic mesh on the rotor surface and the outer domain boundaries is filled with tetrahedrons. As a result, the unstructured mixed-element mesh is build with 2.6 million nodes and 9.9 million elements.

3.3.2. Adaptive mesh for immersed boundary method

To facilitate the mesh adaptation with the metric depending on the distance function the triangulated surface of the propeller is prepossessed and a k-d tree and an octree are built for

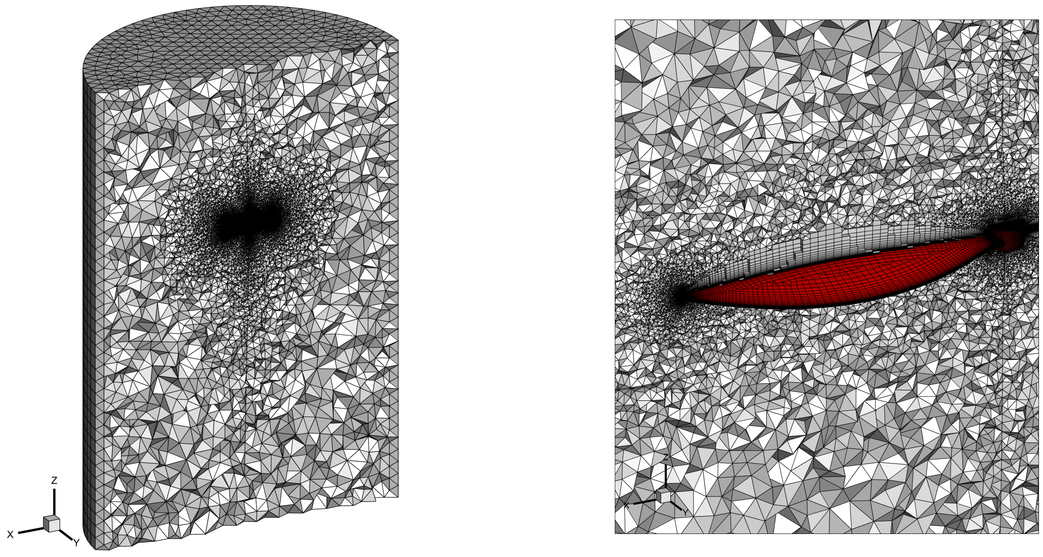


Figure 3. The mesh for the BF approach

fast calculations of the exact and approximate signed distance values, respectively. When possible approximate distance is used, which is calculated by interpolating exact values stored in the octree nodes [22]. In addition to interpolation of the signed distance function the octree is also used to access some shape parameters of the propeller such as principal curvatures, principal normal directions, and distances to internal (Fig. 4 and Fig. 6) and external (Fig. 5 and Fig. 7) medial axes, which are used to define the adaptation control metric. These parameters are found at the vertices of the triangulated surface and then they are extrapolated to the nodes of the interpolation grid (octree) based on the nearest distance. The algorithm for the curvature calculation is described in [23]. The medial axes are approximated using the PowerCrust algorithm [24].

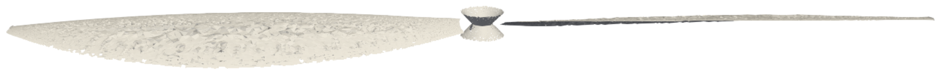


Figure 4. The approximate internal medial axes

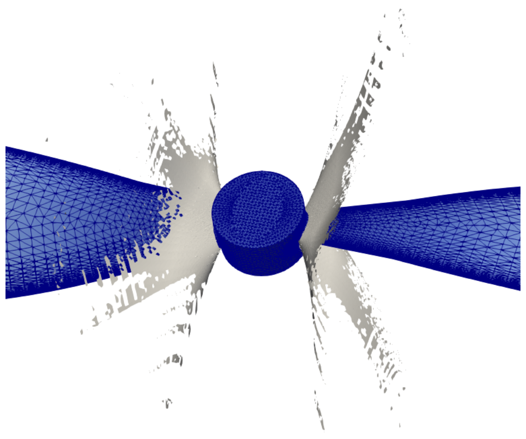


Figure 5. The approximate external medial axes

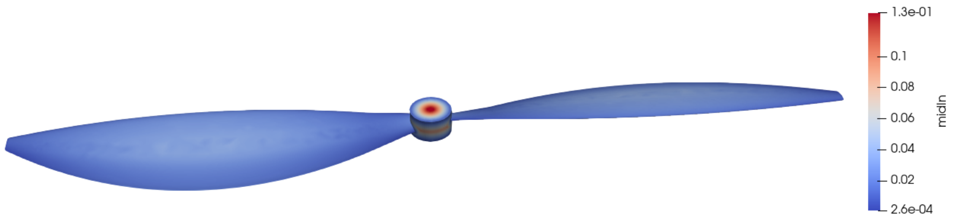


Figure 6. The distances from the surface vertices to the internal medial axes

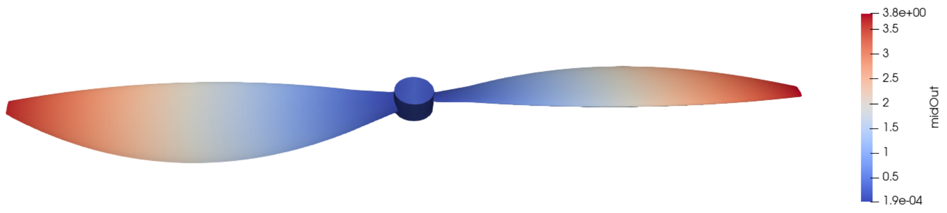


Figure 7. The distances from the surface vertices to the external medial axes

The initial tetrahedral mesh is built in a cylinder-shaped domain with same sizes as in the BF approach with large number of vertices placed near the propeller surface. The total mesh size is 4.4 million vertices with the majority of vertices concentrated in a puck-shaped domain covering the propeller motion region.

The propeller geometry is virtually impossible to reproduce correctly on a simply connected mesh without grid adaptation, since the resolution of thin blade edges requires a significant mesh refinement in the vicinity of the propeller surface.

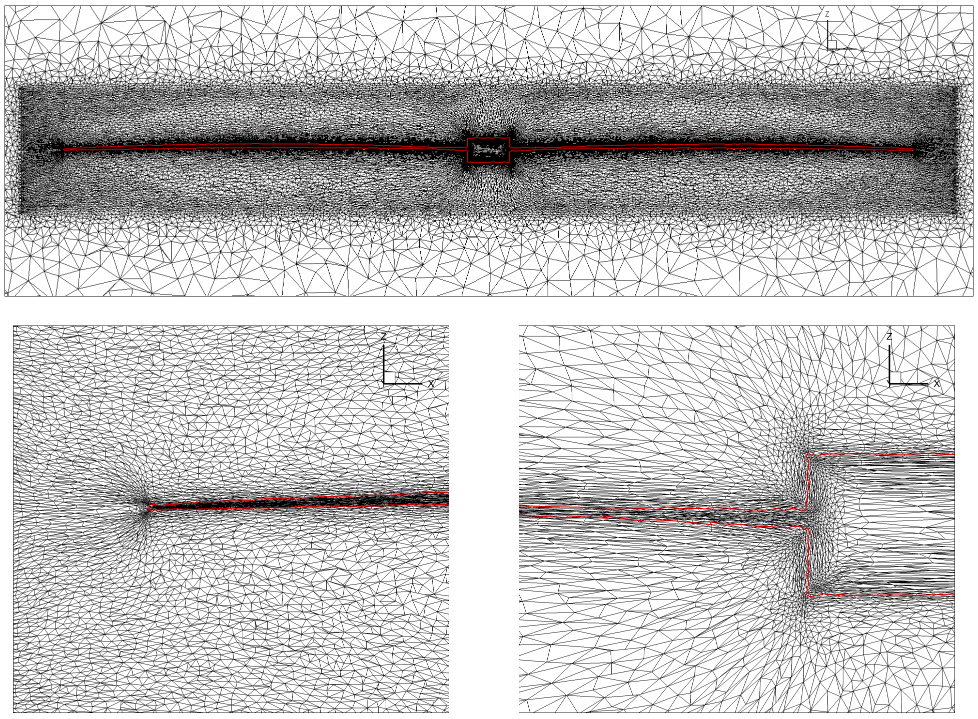


Figure 8. XZ plane section of the DMR-IBM simulation mesh near the propeller (top) and fragments of the adapted mesh near the hub and near the blade tip (bottom)

The mesh redistribution method is then applied to refine the grid to approximately one tenth of the initial mesh size. For the resulting mesh $y^+ \sim 200$, which corresponds to the log-law region. Figure 8 shows a fragment of the adaptive mesh in XZ section and a more detailed image of the mesh near geometry features: the hub and its connection with the blade and the end of the blade. In the presented figures all the points belonging to the body are circled in red. Mesh elements sizes depend on the shape of the body: the mesh is more isotropic near corners and in high curvature regions. Figure 9 presents the body shape in DMR-IBM simulation.

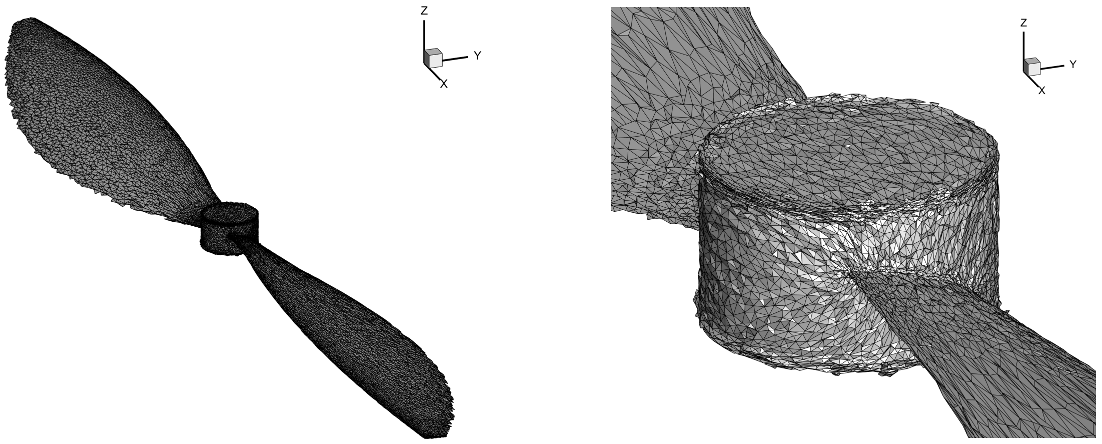


Figure 9. The propeller shape in the DMR-IBM simulation (only the cells with all the vertices inside the body are drawn)

The application of mesh redistribution method is complicated by the presence of thin edges. The distance between the walls of the blade is extremely small and sharp corners produced by these walls are not always well refined using the developed mesh adaptation approach. In the current paper the problem of narrow walls was practically avoided by using a slightly anisotropic initial mesh inside a small cylinder. However, the mesh adaptation algorithm to very narrow parts would need to be further investigated.

3.4. Technology of Acoustics Measurements

To measure and compare the propeller acoustics characteristics in the near field, a set of probes are placed around the rotor. The probes are radially distributed with angular step 10° in plane of rotor rotation (XY) and in XZ plane (Fig. 10). The YZ plane is not considered since, due to the symmetry of the case setup, it is equivalent to the XZ plane.

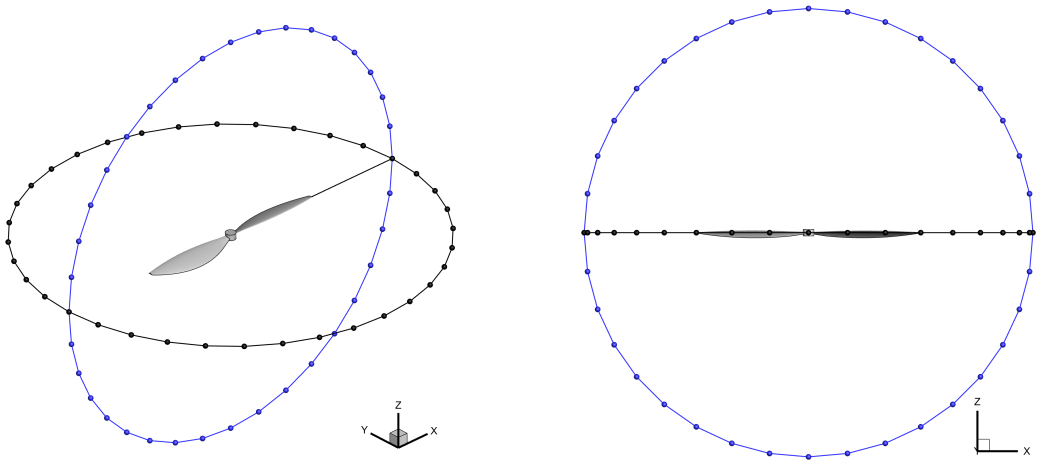


Figure 10. Acoustic probes location: XY plane (black) and XZ plane (blue)

Thus, the following four sets of probes are used: two sets at planes XY, XZ at distance $2R$ from rotor center and two sets at planes XY, XZ at distance $3R$ from rotor center. The azimuthal angle in plane XZ is measured from Ox axis: positive azimuthal angle corresponds to the upper hemisphere and the negative – to the lower (downstream) one.

3.5. Parallel Implementation

All algorithms are implemented in the CFD in-house code NOISEtte. It has MPI+OpenMP parallelization for supercomputers made of multi- and manycore processors. A detailed description of the parallel algorithm is given in [25]. The dynamic mesh adaptation also follows MPI+OpenMP parallelization model. The parallel algorithm uses spacial mesh decomposition. Since degrees of freedom which define mesh deformation are mesh vertices, we build a consistent partitioning of mesh cells and vertices: computational domain is split into connected subdomains consisting of full mesh cells. Mesh vertices belonging to boundaries between subdomains are distributed between subdomains. Parallel BiCGStab-based iterative solver with ILU2 preconditioner is used [21]. Input data for this solver are right hand side partitioned into blocks and sparse matrix partitioned into block rows. Each block precisely corresponds to the partitioning of mesh vertices. The implementation of iterative scheme is based on extended subdomains defining two cell-wide subdomain overlap. We have found that one minimization iteration was enough in order to make mesh precisely follow domain boundaries with prescribed compression. Moreover, it was found that mesh generator can predict mesh deformation with time step exceeding that of RANS solver. In that case on the intermediate time steps mesh is obtained using interpolation technique without any high-load operations. The interpolation sharply improves computational efficiency of mesh deformation solver, which is described in [26].

It should be mentioned that the CFD/RANS simulation of the current problem allows for using the time step exceeding maximal adaptation step. Thus, for the problem under consideration, the adaptation solver was called at each CFD step, while the size of this CFD step was restucted to meet the requirements of the adaptation method. This fact coupled with non-optimal parallel implementation of adaptation solver resulted in rather low efficience of DMR-IBM solution. So far, the current performance of the DMR-IBM method is 8.7 times more expensive than the BF-approach on identically sized meshes with the same time step. The optimized parallel implementation is currently ongoing.

4. Numerical Results

4.1. Validation of BF and DMR-IBM Results on Aerodynamic Characteristics

The flow field in the both DMR-IBM and BF simulations looks similar Fig. 11. There is a jet-type flow downstream the rotor induced by the rotation with the maximum velocity below the blade tips. The tip vortices are well resolved up to 1.5 revolutions and, due to the high mesh resolution in the propeller rotation region in the DMR-IBM case, the tip vortex in the DMR-IBM simulation is resolved better than in the BF case.

In both BF and DMR-IBM simulation the rotor thrust coefficient c_t and power coefficient c_p are compared against the experimental data [11].

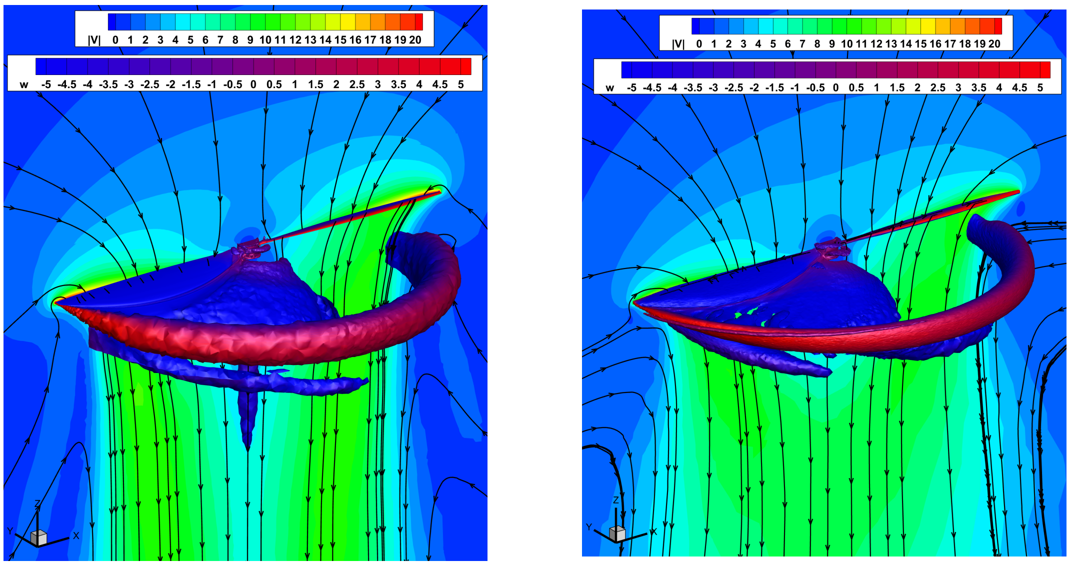

|V| 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
w -5 -4.5 -4 -3.5 -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Z
Y
X
|V| 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
w -5 -4.5 -4 -3.5 -3 -2.5 -2 -1.5 -1 -0.5 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5
Z
Y
X

Figure 11. The flow field: velocity magnitude, streamlines in ZX section with Q-criterion isosurfaces for the BF case (left) and DMR-IBM case (right)

Table 1. Aerodynamic coefficients comparison with the reference data

	Reference	BF	δ, %	DRM-IBM	δ, %
Thrust coefficient c_t	0.1158	0.1178	1.7	0.0947	–18.3
Power coefficient c_p	0.0466	0.0465	–0.2	0.0451	–3.2

The result of comparison of aerodynamic coefficients with the experimental data is presented in the Tab. 1 where the relative difference is calculated as $\delta = \left(c_{t|p} - c_{t|p}^{ref}\right)/c_{t|p}^{ref} \cdot 100\%$. As seen from the Tab. 1, the aerodynamics coefficients are noticeably underestimated in the DMR-IBM simulation. It may be a result of a poor mesh resolution of the boundary layer on the rotor blade surface. Another possible reason of this shortcoming could come from not taking into account the Lagrangian convection in the Brinkman penalization term in (7).

4.2. Comparative Analysis of Acoustic Characteristics

As mentioned above, the primary goal of the work is to study the near-field acoustics generated by the rotating propeller modelled using BF and DMR-IBM approaches. Among the acoustic characteristics, we consider the spectra of pressure pulsation and the directivity diagrams of the pressure pulsation obtained using the above-described probes at the blade passing frequency (BPF). Figure 12 represents the specific spectra of pressure pulsations in two probes – one in the plane of rotation XY with azimuthal position $0°$ (Fig. 12, left) and the other in the plane XZ with azimuthal position $40°$. The presented spectra are built with sampling frequency 6.65 Hz.

As expected, the spectra maxima in both BF and DMR-IBM simulations are reached at the BPF frequency and its multiples. The difference between the BF and DMR-IBM results at first BPF amplitude is about 2.1 dB for the first probe and 1dB for the second one. It is noticeable that the amplitudes of the harmonics that are multiples of the BPF frequencies in the rotation plane obtained in the DMR-IBM simulation are much closer to the results of the BF simulation than in the other azimuthal position.

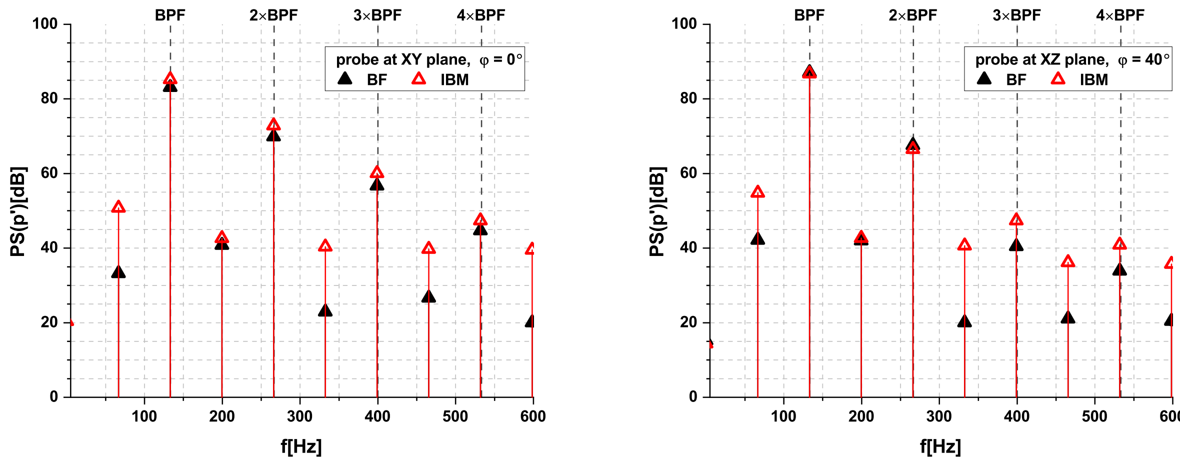


Figure 12. The spectra of pressure pulsation at the probes XY plane azimuth 0°(left), XZ plane azimuth 40° (right)

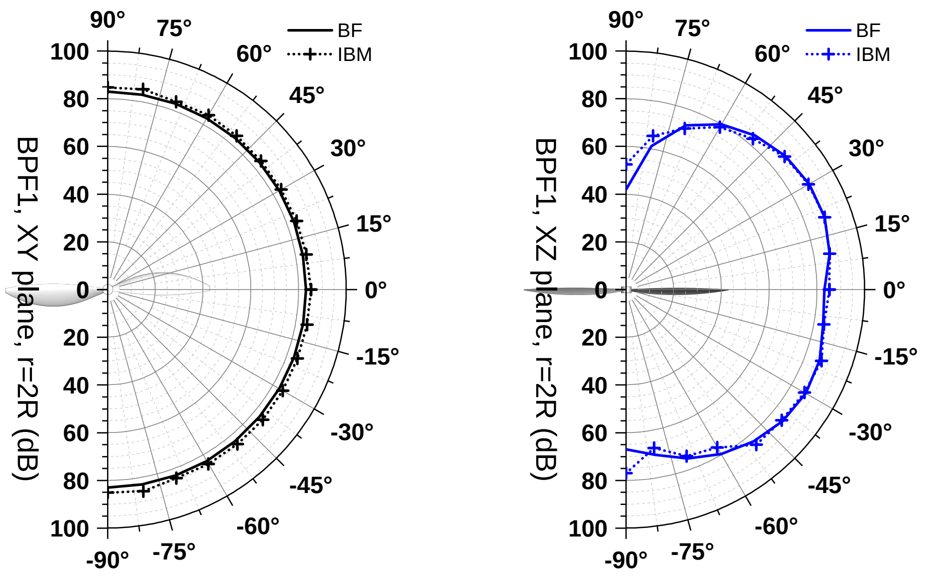


Figure 13. The directivity diagram of the first BPF pressure pulsation at the probes at distance $2R$

In Fig. 13 and Fig. 14 the directivity diagram of the first BPF for the XY and XZ planes are presented for distances $2R$ and $3R$. It is seen that in the plane of rotation the maximum difference between BF and DMR-IBM computation is less than 2.9 dB for $2R$ and 2.2 dB for $3R$. On the other hand, in the XZ plane for azimuthal positions $-75° < \varphi < 75°$ the maximum difference is less than 2 dB for both distances. It should be noted that the pressure pulsation measured at the rotation axes should theoretically be zero. In our case, it is not so (it is about 40–60 dB) solely because of the asymmetric setup. This fact is confirmed by our simulation of a single blade with the periodicity condition in azimuthal direction where it is really zero (see Ref. [14]). Note also that in the DMR-IBM simulation the asymmetric setup is aggravated by the slightly dynamically changing geometry of the blade.

Conclusion

The paper discusses a feasibility of simulating turbulent flow around rotating bodies of complex geometry and the associated acoustics using the developed hybrid dynamic mesh re-

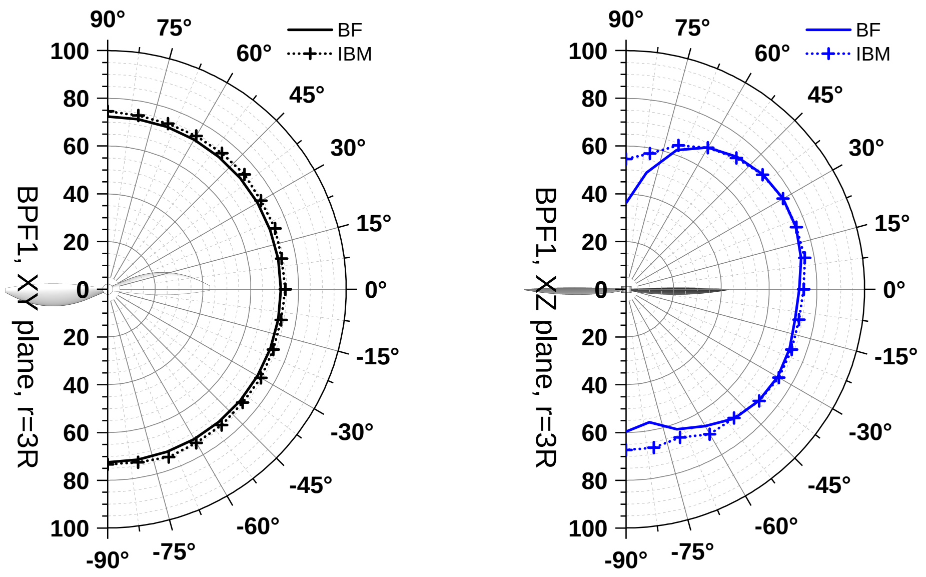


Figure 14. The directivity diagram of the first BPF pressure pulsation at the probes at distance $3R$

distribution – immersed boundary method with mesh metric based on distance function to the obstacle surface. In this paper, we consider the problem of simulation of aeroacoustics of a drone propeller taken as an example of a rotating body of complex geometry. In addition to evaluating the developed method on a representative case, the problem of a single-rotor acoustics has an obvious extension important for a wide range of applications. The current results can be viewed as building block for simulation of the acoustics of multi-rotor machines, the use of which in the daily life is steadily increasing. Although in terms of tonal acoustics the results obtained by the developed hybrid dynamic mesh redistribution – immersed boundary method look promising, the quality of aerodynamic results is not satisfactory and calls for further investigation. One direction that can improve the results of the DMR-IBM simulations to increase the near wall mesh resolution to better represent the immersed solid. Another area of future research is to use wall-functions to decrease the near-wall mesh resolution requirements of the DMR-IBM simulations.

Acknowledgements

The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University and the hybrid supercomputer K60 installed in the Supercomputer Centre of Collective Usage of KIAM RAS. The work was funded by the Russian Science Foundation within Project No. 20-41-09018.

References

1. Abalakin, I.V., Zhdanova, N.S., Kozubskaya, T.K.: Immersed boundary method implemented for the simulation of an external flow on unstructured meshes. Math Models Comput Simul. 8,

219–230 (2016). https://doi.org/10.1134/S2070048216030029

2. Zhdanova, N.S., Gorobets, A.V., Abalakin, I.V.: Supercomputer simulations of fluid-structure interaction problems using an immersed boundary method. Supercomputing Frontiers and Innovations 5(4), 78–82 (2018). https://doi.org/10.14529/jsfi180408

3. Abalakin, I.V., Bahvalov, P.A., Doronina, O.A., *et al.*: Simulating Aerodynanics of a Moving Body Specified by Immersed Boundaries on Dynamically Adaptive Unstructured Meshes. Math. Models Comput. Simul. 11(1), 35–45 (2019). https://doi.org/10.1134/S2070048219010034

4. Abalakin, I.V., Duben, A.P., Zhdanova, N.S., *et al.*: Immersed Boundary Method on Deformable Unstructured Meshes for Airfoil Aeroacoustic Simulation. Comput. Math. and Math. Phys. 59(12), 2046–2059 (2019). https://doi.org/10.1134/S0965542519120029

5. Tsvetkova, V.O., Abalakin, I.V., Bobkov, V.G., *et al.*: Simulation of the Flow near a Rotating Propeller on Adaptive Unstructured Meshes Using the Immersed Boundary Method. Math. Models Comput. Simul. 14(2), 224–240 (2022). https://doi.org/10.1134/S2070048222020168

6. Bobkov, V.G., Vershkov, V.A., Kozubskaya, T.K., Tsvetkova, V.O.: Deformation Technique of Unstructured Mesh Deformation to Find the Aerodynamic Characteristics of Bodies at Small Displacements. Math Models Comput Simul. 13, 986–1001 (2021). https://doi.org/10.1134/S2070048221060028

7. Dervieux, A., Mesri, Y., Alauzet, F., *et al.*: Continuous mesh adaptation models for CFD. Computational fluid dynamics journal 16(4), 346–355 (2008).

8. Tang, J., Cui, P., Li, B., *et al.*: Parallel hybrid mesh adaptation by refinement and coarsening. Graphical Models 111, 101084 (2020). https://doi.org/10.1016/j.gmod.2020.101084

9. Steger, J., Dougherty, F.C., Benek, J.A.: A chimera grid scheme. Advances in Grid Generation 5, 59–69 (1983).

10. Brandt, B.: Small-scale propeller performance at low speed, Master thesis, University of Illinois at Urbana-Champaign, 2005.

11. Brandt, J.B., Selig, M.S.: Propeller performance data at low Reynolds numbers. AIAA Paper 2011-1255 (2011).

12. Bobkov, V., Gorobets, A., Kozubskaya, T., *et al.*: Supercomputer Simulation of Turbulent Flow Around Isolated UAV Rotor and Associated Acoustic Fields. In: Voevodin, V., Sobolev, S. (eds) Supercomputing. RuSCDays 2021. Communications in Computer and Information Science, vol. 1510. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-92864-3_20

13. Spalart, P.R., Allmaras, S.R.: A One-Equation Turbulence Model for Aerodynamic Flows. 30th Aerospace Sciences Meeting & Exhibit, January 6–9, 1992, Reno, NV, AIAA Paper 92-0439. https://doi.org/10.2514/6.1992-439

14. Abalakin, I.V., Anikin, V.A., Bakhvalov, P.A., *et al.*: Numerical Investigation of the Aerodynamic and Acoustical Properties of a Shrouded Rotor. Fluid Dyn. 51(3), 419–433 (2016). https://doi.org/10.1134/S0015462816030145

15. Abalakin, I., Bakhvalov, P., Kozubskaya, T.: Edge-based reconstruction schemes for unstructured tetrahedral meshes. Int. J. Numer. Meth. Fluids 81(6), 331–356 (2016). https://doi.org/10.1002/fld.4187

16. Bakhvalov, P.A., Kozubskaya, T.K.: Construction of edge-based 1-exact schemes for solving the Euler equations on hybrid unstructured meshes. Comput. Math. Math. Phys. 57(4), 680–697 (2017). https://doi.org/10.1134/S0965542517040030

17. Bakhvalov, P., Kozubskaya, T.: EBR-WENO scheme for solving gas dynamics problems with discontinuities on unstructured meshes. Comput. Fluids 157, 312–324 (2017). https://doi.org/10.1016/j.compfluid.2017.09.004

18. Garanzha, V., Kudryavtseva, L.: Hypoelastic Stabilization of Variational Algorithm for Construction of Moving Deforming Meshes. In: Evtushenko Y., Jacimovic M., Khachay M., *et al.* (eds) Optimization and Applications. OPTIMA 2018. Communications in Computer and Information Science, vol. 974, pp. 497–511. Springer (2019). https://doi.org/10.1007/978-3-030-10934-9_35

19. Kozubskaya, T.K., Kudryavtseva, L.N., Tsvetkova, V.O. Anisotropic Adaptation of Moving Unstructured Mesh to Bodies of Complex Shapes Described by an Interpolation Octree. Comput. Math. and Math. Phys. 62, 1590–1601 (2022). https://doi.org/10.1134/S0965542522100074

20. Frazza, L.: 3D anisotropic mesh adaptation for Reynolds Averaged Navier–Stokes simulations. Modeling and Simulation. Sorbonne Université, thesis (2018)

21. Kaporin, I.E., Milyukova, O.Yu.: MPI+OpenMP implementation of the BiCGStab method with explicit preconditioning for the numerical solution of sparse linear systems. Numerical methods and programming 20, 516–527 (2019). https://doi.org/10.26089/NumMet.v20r445

22. Soukov, S.A.: Combined signed distance calculation algorithm for numerical simulation of physical processes and visualization of solid bodies movement. Scientific Visualization 12(5), 86–101 (2020). https://doi.org/10.26583/sv.12.5.08

23. Garimella, R., Swartz, B.: Curvature Estimation for Unstructured Triangulations of Surfaces (2015)

24. Amenta, N., Choi, S., Kolluri, R.K.: The power crust. Proceedings of the sixth ACM symposium on Solid modeling and applications (SMA '01), pp. 249–266 (2001) https://doi.org/10.1145/376957.376986

25. Gorobets, A.V.: Parallel Algorithm of the NOISEtte Code for CFD and CAA Simulations. Lobachevskii Journal of Mathematics 39(4), 524–532 (2018). https://doi.org/10.1134/S1995080218040078

26. Kozubskaya, T., Kudryavtseva, L., Tsvetkova, V.: Unstructured Mesh Adaptation for Moving Bodies in Immersed Boundary Methods. 14th WCCM-ECCOMAS Congress 2020, January 11–15, 2021. https://doi.org/10.23967/wccm-eccomas.2020.353

DOI: 10.14529/jsfi220408

A Numerical Code for a Wide Range of Compressible Flows on Hybrid Computational Architectures

Anton A. Shershnev[1], *Alexey N. Kudryavtsev*[1], *Alexander V. Kashkovsky*[1], *Georgy V. Shoev*[1], *Semyon P. Borisov*[1], *Timofey Yu. Shkredov*[1], *Danila P. Polevshchikov*[1], *Alexey A. Korolev*[1], *Dmitry V. Khotyanovsky*[1], *Yulia V. Kratova*[1]

The major points in the development of the parallel multiplatform multipurpose numerical code solving the full unsteady Navier–Stokes equations are presented. The developed code is primarily designed for running on multi-GPU computational devices but can also be used on traditional multicore CPUs and even on manycore processors such as Intel Xeon Phi. Physical models include calorically perfect inert gas, single- and multi-temperature approaches for chemically reactive flows and an Euler–Euler model for gas-particle suspensions. Main details of the implementation are described. Shock capturing TVD and WENO schemes in general curvilinear coordinates are used for spatial approximation. Explicit, semi-implicit and fully implicit schemes are employed for advancing solution in time. The code is written in C++ with CUDA API and opensource libraries, such as MPI, zlib and VTK. A few examples of numerical simulations are briefly described to provide general idea of the numerical code capabilities. They include a supersonic flow past a wedge, a jet exhausting from a square nozzle, a heavy gas bubble descending in a lighter medium and a heterogeneous detonation in gas-particle suspension.

Keywords: Navier–Stokes equations, numerical simulation, compressible flows, DNS, thermochemical non-equilibrium, GPGPU, CUDA.

Introduction

Currently numerical simulation has become one of the main tools for conducting scientific research. Fluid dynamics of compressible flows is one of the areas where numerical tools are particularly ubiquitous. However, the compressibility of the flow is associated with the presence of shock waves and other hydrodynamic discontinuities and, therefore, requires rather sophisticated and computationally expensive shock-capturing numerical schemes. Additionally, modern fundamental problems of interest are associated with a wide range of considered scales, including transitional to turbulent and fully turbulent flows, chemically reactive flows, and multiphase flows. And despite the fact of extensive use of parallel computations, such simulations can take hundreds of hours of computational time even on high-performance clusters.

There is a large number of high-performance codes developed for numerical simulations of compressible flows. In particular, in Russia well-known research codes are NERAT [1], Noisette [2], EWT-TsAGI [3], NTS [4], Flowmodellium [5], VP2/3 [6], Jet3D [7], SINF [8] as well as more industry-oriented codes LOGOS [9] and FlowVision [10].

These and many other codes implement different physical models for laminar, turbulent, inert and chemically reactive flows, use different computational meshes and numerical schemes and parallelization techniques. The latter include traditional MPI-based approach, more recent OpenMP application programming interface and even rather exotic OpenCL framework. For any of the numerical tools mentioned above, physical models, numerical techniques, parallelization

[1]Khristianovich Institute of Theoretical and Applied Mechanics, Siberian Branch of Russian Academy of Sciences, Novosibirsk, Russian Federation

methods and programming solutions used in a code reflect the research fields of interest, personal experience and specific needs of its developers and users.

One of the reasonable ways to increase efficiency and significantly reduce computational wall-clock time is to employ modern general-purpose graphics processing units (GPGPU) with high computational efficiency and data throughput (see, e.g., [11–13]). The present paper summarizes the main details of the HyCFS-R numerical code [14–17] developed at the Laboratory of Computational Aerodynamics of Khristianovich Institute of Theoretical and Applied Mechanics and designed to run on hybrid CPU/GPU computational systems. The code solves the full unsteady Navier–Stokes equations using modern shock capturing high-order schemes. In this paper we describe governing equations and discuss some details of the implementation including the problems specific for hybrid and heterogeneous computations.

The rest of the paper is organized as follows: Section 1 describes governing equations and numerical techniques of their solution, in Section 2 the details of the program implementation are given, including general architecture, data structures etc., and in the last section examples of performed numerical simulations are presented.

1. Governing Equations

The code solves the full 3D unsteady Navier–Stokes equations for inert and chemically reactive multispecies mixtures on a structured mesh in general curvilinear coordinates:

$$\frac{\partial \mathbf{Q}}{\partial t} + \frac{\partial \mathbf{F}_x}{\partial x} + \frac{\partial \mathbf{F}_y}{\partial y} + \frac{\partial \mathbf{F}_z}{\partial z} = \frac{\partial \mathbf{F}_{v,x}}{\partial x} + \frac{\partial \mathbf{F}_{v,y}}{\partial y} + \frac{\partial \mathbf{F}_{v,z}}{\partial z} + \mathbf{S}, \tag{1}$$

$$\mathbf{Q} = (\rho u, \rho v, \rho w, E, \rho, \rho_1, \ldots, \rho_N,\ \rho_{i_1} E_{v_1}, \ldots, \rho_{i_M} E_{v_M},\ \rho\tilde{\nu},\ \rho_p u_p, \rho_p v_p, \rho_p w_p, E_p, \rho_p)^T, \tag{2}$$

where $\mathbf{Q}$ is the pseudo-vector of conservative variables, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$ are convective fluxes, $\mathbf{F}_v$, $\mathbf{G}_v$, $\mathbf{H}_v$ are viscous fluxes, and $\mathbf{S}$ denotes source terms.

Options for physical models in numerical simulation include:

- A mixture of N calorically perfect gases for laminar and turbulent regimes. Specific heats for each species are constant, i.e., do not depend on the temperature and are calculated in accordance with the number of the molecular degrees of freedom.
- A thermally perfect gas mixture with the single-temperature model for finite-rate chemical kinetics. In this case gas mixture thermodynamic properties are calculated using widely employed approach based on the polynomial approximations with coefficients taken from one of the available sources: the database by Alexander Burcat [18], the NASA database [19] or approximations used by Gupta [20].
- A multi-temperature model with equations for the vibrational energies of molecular species and finite-rate chemical kinetics. In this case, we assume that thermodynamic properties of the translational-rotational mode of the species are given by the relations for the calorically perfect gas, so the specific heats are constant and depend only on the number of degrees of freedom. The rate of translational-vibrational exchange is calculated using either the classical Landau–Teller equation [21], or new model [22], which was obtained using the rigorous methods of the kinetic theory of gases.
- An Euler–Euler model for the chemically reactive gas-particle suspension. The model is based on the concept of interpenetrating continua. The governing equations that describe

the detonation flow in polydisperse gas suspensions of aluminum particles are based on the works of Fedorov and Khmel [23–25].

The pressure of the gas mixture in all cases is calculated using Dalton's law from the partial pressures of components. The transport coefficients of the gas mixture are calculated from those of the individual species using the Wilke mixing rule. Dynamic viscosities of individual species can be calculated using the power law, the Sutherland law or via Ω-integral approximations of the kinetic theory. The thermal conductivities are given by the Eucken formula with Hirschfelder correction [26]. Finally, the option of adding an external field of gravitational force is also available in the code.

2. Numerical Techniques

All spatial approximations in the code are based on the shock capturing TVD (total variation diminishing) and WENO (weighted essentially non-oscillatory) schemes formulated in general curvilinear coordinates. The high-order TVD schemes are implemented using MUSCL (Monotonic Upstream-centered Scheme for Conservation Laws) approach initiated in [28, 29]. More specifically we use the 2nd/3rd order MUSCL formula proposed in [30] along with the **minmod** slope limiter to reconstruct flow variables on cell boundaries from their cell-centered values. Numerical fluxes are calculated using one of the approximate Riemann solvers implemented in the code. The list of available Riemann solvers include HLLE [31] (Harten–Lax–van Leer–Einfeldt), HLLC (Harten–Lax–van Leer–Contact) [32] with an estimate for the contact-wave speed from [33], Roe [34], and a few solvers from the AUSM (Advection Upstream Splitting Method) family: AUSM-Van Leer [35, 36], AUSM-up [39], AUSM+(P) [37], and AUSMPW+ [38]. Alternatively, for the simulation of a calorically perfect gas the WENO scheme of the 5th order by Jiang and Shu [27] with either local or global Lax–Friedrichs flux splitting can be used. Convective terms for the disperse phase continuum are calculated using the MUSCL reconstruction combined with the dusty gas Riemann solver from [40]. Diffusive terms are calculated using the central differences of the 2nd order.

The solution is advanced in time using explicit, semi-implicit or implicit schemes. Explicit time integration is based on the so-called Runge–Kutta TVD schemes of the 1st through the 3rd order [41], or the low-storage Runge–Kutta–Gill scheme of the 4th order [42]. In the additive semi-implicit Runge–Kutta scheme of the 2nd order (ASIRK2C) [43] stiff chemical source terms are evaluated implicitly while for the convective terms more efficient and simpler explicit methods are used. The implicit DPLUR (Data Parallel Lower-Upper Relaxation) scheme [44] available for non-reacting flows allows one to integrate equations in fully implicit manner in parallel computations.

2.1. Boundary Conditions

The imposition of boundary conditions in the HyCFS-R code is implemented using the so-called ghost cells technique. The computational domain is surrounded by three rows of virtual cells, in which flowfield variables are calculated according to the type of boundary condition. Generally speaking, the number of the ghost-cell rows depends on the size of the stencil, but HyCFS-R always uses three layers, corresponding to the 7-point stencil in high-order schemes.

The following boundary types are implemented in HyCFS-R: supersonic inflow and outflow, pressure-constant inlet and outlet, periodic boundaries, and solid boundaries, such as inviscid, no-slip isothermal and no-slip adiabatic walls. Additionally, special boundary conditions for superposing disturbances onto the flow are implemented: in the form of eigenfunctions of the linear stability problem, white gaussian noise and periodic thermal fluctuations. The last three conditions are typically used for flow forcing when simulating transition to turbulence in shear flows.

It should be mentioned, that the code employs specific scheme of boundary conditions imposition. In most papers this stage of numerical algorithm is described very briefly and it is usually implied that values in ghost cells are recalculated at the beginning of the new time step based on the values in the internal cells. In HyCFS-R boundary conditions for inviscid and viscous numerical fluxes are imposed separately, to ensure conservation of mass at the solid boundaries.

3. Details of the Program Implementation

The program is written in C++ and uses only open-source tools, libraries and technologies and in-house utilities, such as GNU/Linux, GCC, OpenMP, MPI, zlib, VTK, to avoid vendor-lock situations often encountered when using commercial solutions. The primary computational platform of the code are hybrid CPU/GPU clusters and all numerical procedures in the code are implemented with SIMD (single instruction multiple data) execution in mind. It should be noted that computations of numerical fluxes and source terms in the Navier–Stokes equations mostly consist of uniform-length loops with fixed-size stencils and small number of conditional operators. Such computations match the aforementioned SIMD architecture quite well and straightforward implementation of numerical routines provides sufficient efficiency.

For the maximum flexibility of the code it can be compiled to run on conventional multicore CPUs using a special compile-time wrapper for the C preprocessor, which converts GPU-specific entities to OpenMP syntax. In this approach each OpenMP thread is interpreted as a GPU thread. Just as GPU kernel function is executed by each CUDA thread, the converted CPU version of the kernel is executed on each OpenMP thread. The only difference is that OpenMP threads are not forced on a hardware level to be executed synchronously, i.e., in a SIMD manner. However, it is not important in our computations. This technique allows one to significantly increase the number of supported computational architectures and also simplifies debugging of the code when using conventional C++ development tools, such as **GDB debugger, Valgrind** framework for code dynamic analysis and so on.

3.1. Input and Output Files

Internal formats are used for all input data: mesh, flowfields, chemical mechanisms description and general configuration options. Mesh and flowfields are stored in simple ASCII or binary files, compressed with zlib library to reduce storage space. General configuration is processed using a special configuration reader, also developed at the Laboratory of Computational Aerodynamics at ITAM SB RAS. The reader is based on the “key-value” scheme. The parameters are divided into groups corresponding to its type and/or parent data structure. The reader also supports `#include` directives, allowing to directly insert contents of one file into another, like in a typical C/C++ code, for additional flexibility of the configuration. The reader stores both the list of all keys supported by the solver and the list of the keys actually read from the input files and checks if the parameter values fall out of the required range. Preprocessing tools for importing files from third-party programs are also included in the code, e.g., tools for reading the description of chemical reaction mechanisms in CHEMKIN format, for the mesh in FlowLogic format and some others.

Postprocessing utilities implemented in the code allow one to export flowfields in ASCII and binary Tecplot format, and legacy ASCII VTK formats, calculate integrated and mean values of any variable, extract distributions from arbitrary line segment, and calculate integrated and distributed aerodynamic characteristics on the solid walls.

Most aerodynamic characteristics are based on the following normalized and integrated quantities at the solid wall surface: the pressure p_w, the frictional stress $\tau_w = \mu_w(\partial u_s/\partial n)$ and the heat flux $q_w = \kappa_w(\partial T/\partial n)$. Here $\mathbf{n}$ and $\mathbf{s}$ are the directions normal and tangent to the wall, respectively. Normal and tangent components of velocity are calculated from Cartesian velocity components stored in the flowfields. Then derivatives along the normal and tangent directions are transformed to the derivatives in general curvilinear coordinates. The quantities, such as velocity $\mathbf{u}$, dynamic viscosity μ, heat conductivity κ and Jacobian of transformation $\mathcal{J}$ between Cartesian and curvilinear coordinates are extrapolated to the wall with the 1st or 2nd order of accuracy. After calculation of the distributed characteristics the integrated characteristics are obtained by summation of local values multiplied by the surface element areas.

3.2. Data Structures

3.2.1. cContainer class

Because the HyCFS-R code contains various physical models with different number of equations, which can be switched at runtime, the data structures used for flowfields storage should also have an adjustable list of variables and means to access these variables in a convenient way. Conventional containers from C++ STL (Standard Template Library) cannot be used on GPU devices, so one has to implement custom container for the flowfields. The HyCFS-R uses a class **cContainer** with a simple array of floating point values at its core, extended with a few extra features, such as a “CPU/GPU” flag (indicating in which memory the data is allocated), a list of variables names, functions for accessing values by the number of cell and the name or the number variable, possibility to store in a cell not only a list, but also a full matrix of parameters. The latter is useful when dealing with various Jacobians and other matrices used in implicit methods.

3.2.2. Mem class

When developing large complex codes for the numerical simulation, the programmers have to change the list of parameters of various functions on a regular basis. Additional parameters are introduced when programming new numerical methods, physical models or new implementations of existing functions. Naturally, any change in the list will require revisions in every occurrence of the function call. To avoid or at least reduce the amount of this tedious work, in the HyCFS-R we store all variables and pointers to arrays in a single structure **Mem**, instances of which are stored in both the CPU and GPU memories. We avoid the word ‘copies’, because for each Mem class member we specify explicitly in which memory (CPU or GPU) it will be allocated. It allows us to minimize the array duplication.

A pointer to **Mem** structure passed to the functions as an argument provides the access to all data available in a computation. One of the advantages of this approach is that most of the memory allocation routines are limited to methods of a single class, allowing to simplify the control of memory consumption and prevention of memory leaks and corruption.

When using domain decomposition, each partition has its own dedicated pair of CPU and GPU instance of Mem class.

3.2.3. BSegment and condition classes

When imposing boundary conditions, the boundary of the domain is divided into partitions we call *segments*. They are rectangular in the $\{\xi, \eta, \zeta\}$-space and, generally speaking, can overlap. Geometrical parameters of each segment and the type of boundary conditions assigned to this segment are stored as object of the dedicated class **BSegment**. Physical parameters of boundary conditions are stored as separate **Condition** objects, referenced by one of the segments. The conditions contain wall temperature, free-stream flow parameters and so on. This mechanism of splitting of geometrical and physical parameters of a boundary condition allows one to re-use the segments for organization of inter-block exchange in a multiblock structured mesh. For this case the segments also store a multiblock interface identificator and exchange buffers of an appropriate size. Figure 1 shows a schematical example of domain boundary divided into segments, including a multiblock partition.

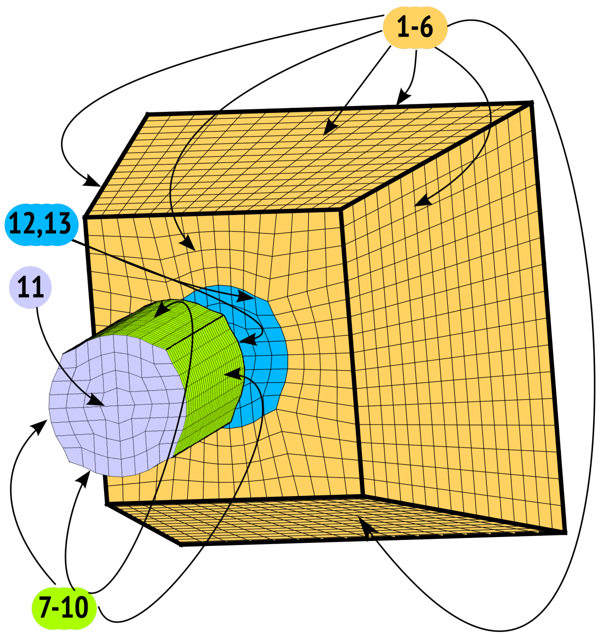


Figure 1. Schematic of boundary segments

3.2.4. Element and reaction classes

The physical and chemical properties of the species and descriptions of chemical reactions are stored in the dedicated structures. And since CUDA platform puts certain limitations on C++ features, inheritance is used mostly to separate subsets of parameters describing gas and condensed phases. So, the hierarchy of the classes is as follows: the classes **Gas** and **Condensed** are both inherited from the basic class **BaseElement** and the classes **GasReaction** and **HeterogenousReaction** are both inherited from the class **BaseReaction**.

3.3. Parallelization

In the most general case the HyCFS-R uses a multi-level parallelization based on the CUDA API, OpenMP threads and the MPI protocol. There are 2 variations of the numerical algorithm: one is used for domains with simple geometries and regular meshes and the other is used for complex domains with structured multiblock meshes. In both cases the whole computational domain is divided into a number of partitions, corresponding to the number of used GPUs. Each GPU performs computations in cells of its sub-domain and after each step exchanges data with neighboring sub-domains. GPUs on one computational node are controlled using OpenMP threads and exchanges are made via the CPU memory. Inter-nodal exchanges are performed using the MPI protocol.

The difference is that in the simple variant the domain is partitioned along the k-index (ζ-axis) as shown in Fig. 2. As a result, neighboring previous and subsequent partitions are unambiguously defined from the current partition index. Another benefit of this scheme is that cells with the same index on the ζ-axis are in a continuous memory segment and do not require extra buffers for exchange.

As for the multiblock exchange, it requires not only the description of every block-block connection via the segment mechanism, but also a special configuration file assigning each block to a GPU device and to an MPI node.

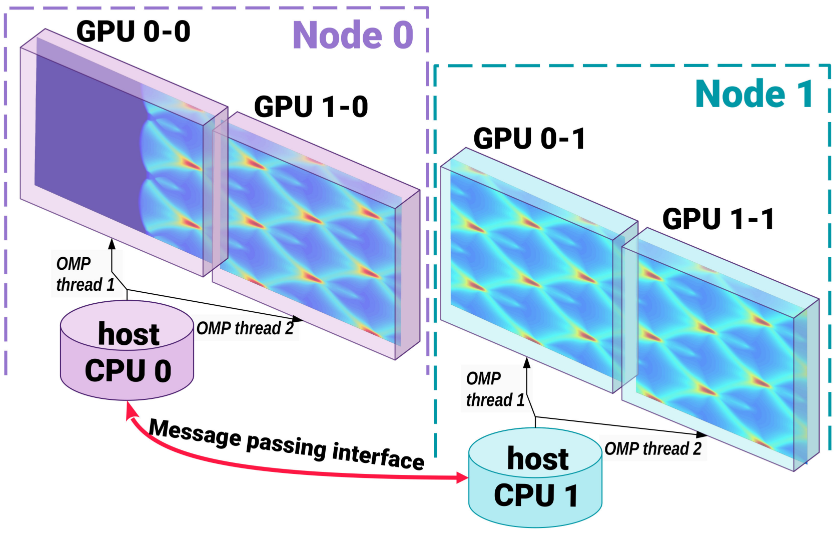


Figure 2. Scheme of multi-level parallelization

The multilevel approach described above allows one a hybrid parallelization when both GPUs and multicore CPUs are used as computational units. Hybridization is achieved by employing OpenMP for multicore CPUs in the same way as CUDA threads are used on GPUs. The source code is the same on all platforms, which, combined with a proper compile-time wrapper, ensures that all computations are performed in the same way. Execution and data exchange are done using the MPI library. Figure 3 shows an example of devices configuration in a hybrid computation. In this example the domain is divided into 8 partitions, represented by gold and green blocks at the bottom of the scheme, where the gold partitions are computed on GPU, and green are computed on CPU, respectively. The configuration consists of 2 nodes with single 10-core CPU and 2 GPUs on each node. The MPI processes with ranks 0 and 1 control the GPUs and the rest of processes control CPU computations with OpenMP threads. This example illustrates a typical situation when using hybrid CPU/GPU clusters, where the number of CPU cores is significantly higher than the number of GPU devices.

In practice such hybrid computations are rarely performed because most of the popular job schedulers lack features to manage resources for them. And it also should be noted that complex computations require to balance the load of a computational devices, e.g., by matching the size of each partition to the performance of the device.

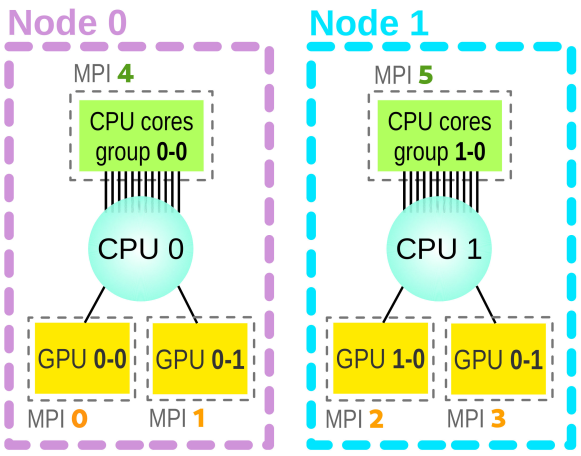


Figure 3. Example of hybrid CPU-GPU parallelization scheme. 6 MPI, 8 domains: 4 GPU + 4×4 OpenMP(CPU)

3.3.1. Service and debugging options

Development of any CFD program is associated with thorough testing for correctness, accuracy and performance of the code. For these purposes, all calls of computational routines are surrounded by wall-time measuring functions and followed by special inspecting routines, which allow one to check all affected flowfields for any changes.

The timers are implemented in a simple class based on the standard **gettimeofday** function and **std::map<string,double>** container. They provide simple statistics for the execution time between two given lines of code and can be switched off at runtime using the configuration file. This statistics is sufficient for initial assessment of hot spots and bottlenecks in the code. A more detailed analysis can be done using external tools such as **valgrind/kcachegrind**, **gperftools**, **Intel VTune** and so on.

As for the inspection, it is enabled at compile time and is only used in the debug-build of the code. This build uses not only inspection routines but also specific build flags for the detection of floating point exceptions, enabling faster search for the problematic fragments.

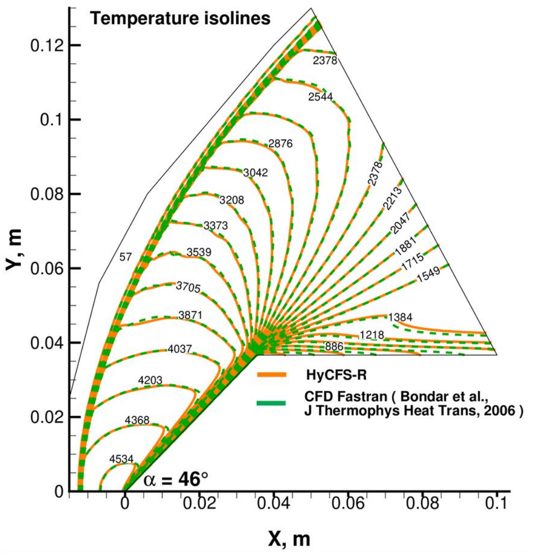
4. Some Examples of Numerical Simulations

This section contains a brief description of a few numerical studies performed using the developed numerical code, which illustrate some typical problems of the compressible flow gas dynamics and the capabilities of the code.

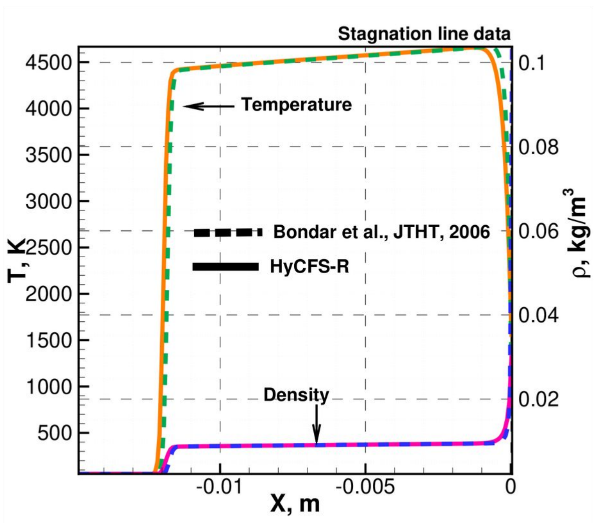
4.1. Supersonic Flow Past a Wedge

A supersonic flow of pure argon past a 46° wedge for the conditions of experiment by Hornung and Smith [47] was simulated numerically and compared with the data from [45]. The mesh size is identical to that used for Navier–Stokes computations in [45], namely 360×200 cells.

Results are shown in Fig. 4, where the comparison of temperature and density distributions are depicted. As can be seen, the HyCFS-R solution is in excellent agreement with the data from [45] in terms of the shock wave stand-off distance. This test verifies spatial approximation and boundary conditions at the solid walls.



(a) Temperature isolines



(b) Temperature and density distributions along the stagnation line

Figure 4. Supersonic argon flow past a 46 degree wedge

4.2. Jet Exhausting from a Square Nozzle

A 3D numerical simulation of the development of instabilities was carried out for a supersonic perfectly-expanded jet exhausting from a nozzle of the square cross-section into an ambient stream. The simulations were performed for the jet Mach number $M_j = 2.5$ and the co-flow Mach number $M_a = 1.5$ at the Reynolds number Re = 5000. The computational domain had the hexahedral shape. The computational grid was refined around the jet core and its near field. The computations were carried out on the grid $1152 \times 330 \times 330 \approx 125$ million cells. The jet instability was excited by superimposing disturbances in the form of a white Gaussian noise onto the main flow velocity components.

Figure 5 shows isosurfaces and contours of density in a central cross-section. As can be clearly seen, at the initial stage, near the inflow boundary, the instability develops in the form of a typical chain of the Kelvin–Helmholtz vortices. This development of instability is quite expected for the considered values of the ambient and jet flow Mach numbers. Downstream, the vortex motion starts to dominate the entire jet core. Overall, the results of these numerical simulations allowed us to identify specific features of the instability development in a supersonic rectangular jet with a supersonic co-flow.

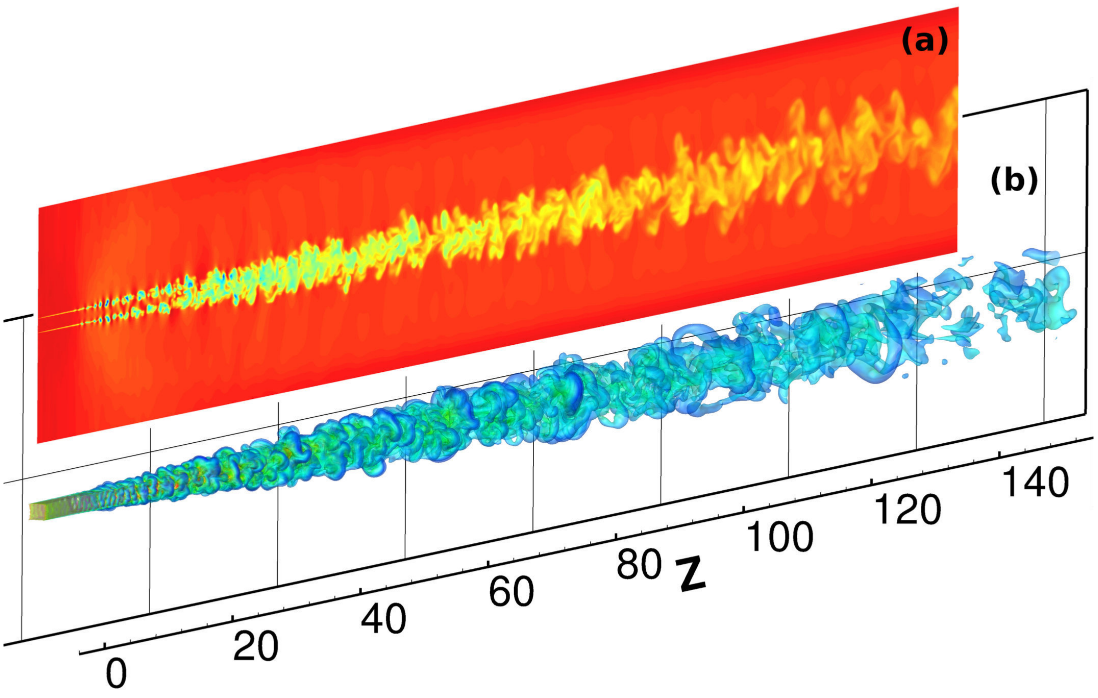


Figure 5. Rectangular jet with Mach number $M_j = 2.5$ exhausting into ambient coflow with Mach number $M_a = 1.5$. Density contours in middle cross-section (a) and density isolines colored with local Mach number values (b)

4.3. Instability of a Gas Bubble in a Gravitational Field

Numerical simulations of the instability of cylindrical and spherical volumes of argon surrounded by helium in the field of gravity were carried out. The volume diameter was set equal to 20 m, the initial temperature in all domain T = 300K. The gravitational acceleration was equal to $g = 1000$ m/s^2. The barotropic distribution of density and pressure was set both inside the volume and in the surrounding light gas. The pressure at the bottom wall was about 1 Pa.

Three cases were considered and 2D, axisymmetric, and 3D computations were carried out for the following domain and mesh sizes. The 100×200 m domain with 800×1600 grid was used for two-dimensional computation, the domain was halved along the axis of symmetry in axisymmetric computation down to 50×200 m with mesh containing 400×1600 cells. In 3D simulation $100\times100\times200$ m domain with $280\times280\times560 \approx 43.9$ million cells was used.

The results of various computations were compared with each other (see Fig. 6). The volume in all simulations changed shape from round in cross section becoming close to semicircular, and then transforming into a crescent-like structure, with cusps pointing upwards. These cusps begin to twist, a secondary instability develops on them. At first the cusps and then entire bubble start to break down, mixing with the ambient gas. In all geometries the instability emergence and development as well as the breakdown process have their own characteristics. The fastest mixing and the breakdown of the heavy gas bubble was observed in the 3D simulation while the mixing process in the 2D simulation was the slowest, because of its cylindrical shape.

4.4. Heterogeneous Detonation in Al/O_2 Gas-particle Suspension

In this test case a detonation wave propagating through a channel filled with oxygen and 10 μm aluminum particles suspension is investigated numerically. Gas phase parameters corre-

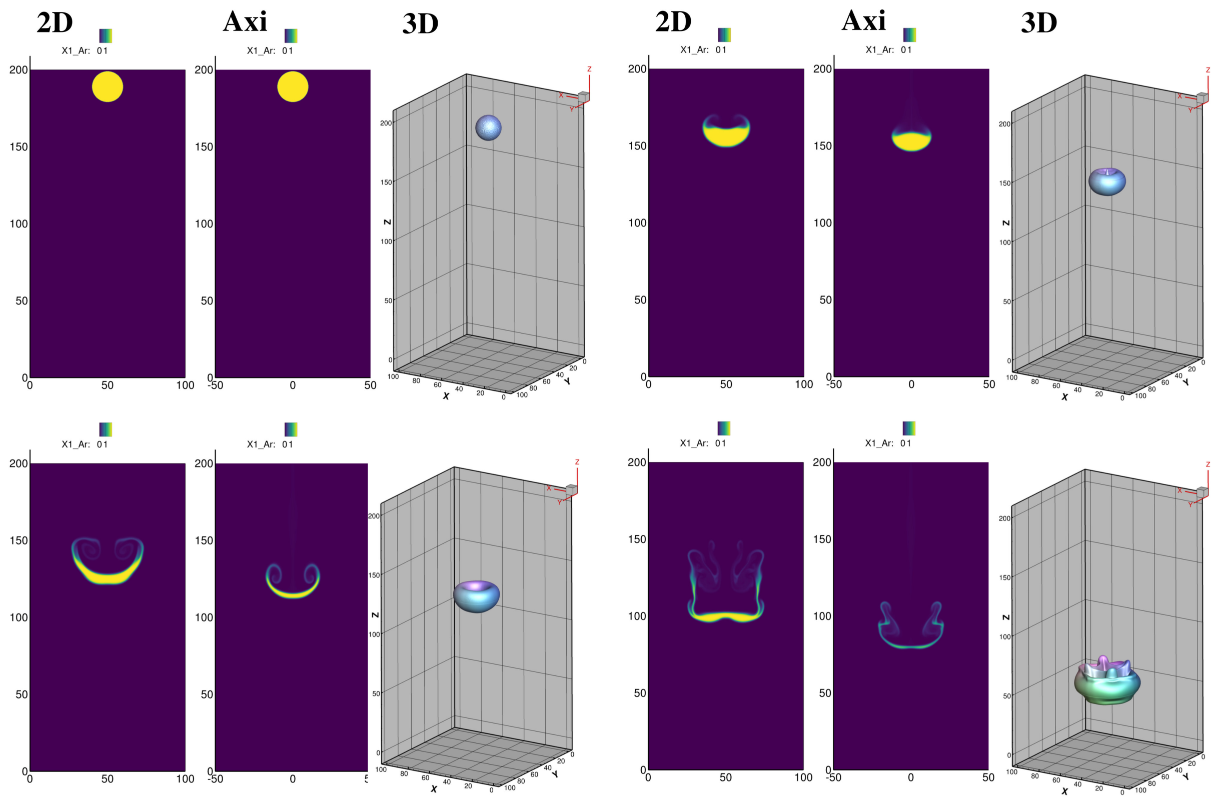
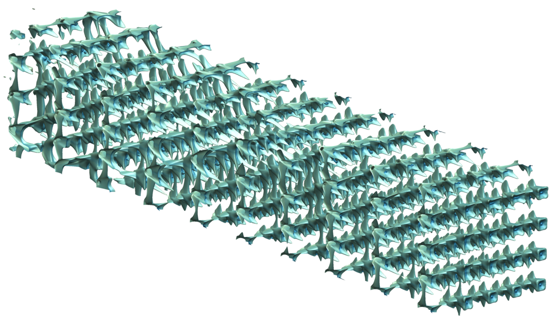


Figure 6. Instability of an argon bubble in helium. Argon molar fraction contours in 2D, axisymmetric and 3D NS computations shown at different time instants

spond to the standard atmospheric conditions with the pressure of 101325 Pa and the density of 1.28 kg/m^3. Detonation is initiated by a localized ignition at the left boundary of the computational domain. The 22.5 m $\times$ 1 m $\times$ 1 m channel was simulated as 3 separate 7.5 m long sections each with the spatial resolution of 2400 $\times$ 350 $\times$ 350 $\approx$ 300 million of cells. The computations were carried out using 8 Nvidia Tesla V100 GPUs and took 96 hours of wall-time.

Figure 7b shows the history of maximum pressure in the form of isosurfaces and distributions on the channel lateral sides, imitating the soot-covered foils from the real experiment. The latter also illustrate the process of transformation of the initial blast wave into a regular cellular structure. Additional code features were used to measure the speed of detonation wave front and also capture its shape.


(a) Isosurface in the flowfield of history of maximum pressure

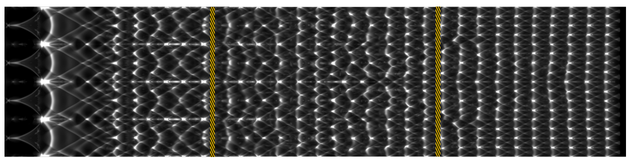

(b) Numerical “soot-foil” records on the channel walls

Figure 7. Heterogeneous detonation in a rectangular channel filled with Al/O$_2$ gas-particle suspension

Conclusion

The HyCFS-R numerical code for solving the compressible Navier–Stokes equations on parallel hybrid computational systems equipped with multiple GPUs or manycore co-processors has been developed. The developed code can be used to simulate gaseous flows including flows of thermally non-equilibrium and chemically reacting gases and gas mixtures as well as multiphase gas-particle suspensions. Modern shock-capturing TVD and WENO schemes on structured muiltiblock meshes are used for spatial approximation, while the time stepping is performed with explicit RK TVD, semi-implicit ASIRK2c and implicit DPLUR schemes. The descriptions of flowfields, boundaries and boundary conditions, physical and thermodynamic properties of the gases, chemical kinetics, are stored as simple C++ classes and structures compatible with CUDA API. Multilevel parallelization of the HyCFS-R code is achieved via a set of techniques, namely MPI, OpenMP and CUDA technologies. Third-party software tools such as **GNU Data Debugger**, **valgrind/kcachegrind**, and **Intel VTune** are used for debugging and code optimization, **zlib** library is used for flowfields data compression and **VTK** library is used for data visualization.

The HyCFS-R code has been employed to simulate a number of subsonic and supersonic flows: a flow past a wedge, a jet exhausting from a square nozzle, a bubble of heavy gas descending in a lighter medium, a heterogeneous detonation in a rectangular channel. Numerical grids containing up to 300 million of cells has been used. The results of numerical simulations presented in the paper give evidence that the developed code is capable to efficiently simulate a wide range of compressible flows.

Acknowledgements

This work was supported by the Russian Science Foundation, grant No. 18-08-01442-П.

References

1. Surzhikov, S.T.: Radiation aerothermodynamics of the Stardust space vehicle. Journal of Applied Mathematics and Mechanics 80(1), 44–56 (2016). `https://doi.org/10.1016/j.jappmathmech.2016.05.008`

2. Abalakin, I.V., Bakhvalov, P.A., Gorobets, A.V., *et al.*: Parallel software package NOISETTE for large-scale computations in fluid dynamics and aeroacoustics. Num. Meth. Prog. 13(3), 110–125 (2012).

3. Neyland, V.Y., Bosnyakov, S.M., Glazkov, S.A., *et al.*: Conception of electronic wind tunnel and first results of its implementation. Progress in Aerospace Sciences 37(2), 121–145 (2001). `https://doi.org/10.1016/S0376-0421(00)00013-0`

4. Shur, M., Strelets, M., Travin, A.: High-order implicit multi-block Navier–Stokes code: Ten years experience of application to RANS/DES/LES/DNS of turbulent flows. `https://cfd.spbstu.ru/agarbaruk/doc/NTS_code.pdf`

5. Petrov, M.N., Tambova, A.A., Titarev, V.A., *et al.*: FlowModellium Software Package for Calculating High-Speed Flows of Compressible Fluid. Comput. Math. and Math. Phys. 58, 1865–1886 (2018). `https://doi.org/10.1134/S0965542518110118`

6. Isaev, S.A., Baranov, P.A., Usachov, A.E.: Multiblock computational technologies in the VP2/3 Package on aerothermodynamics. LAP LAMBERT Academic Publishing, Saarbrucken (2013).

7. Lebedev, A.B., Lyubimov, D.A., Maslov, V.P., *et al.*: The prediction of three-dimensional jet flows for noise applications. AIAA Paper no. 2002-2422 (2002). `https://doi.org/10.2514/6.2002-2422`

8. Smirnov, E.M., Zajtsev, D.K.: The finite-volume method in application to complex-geometry fluid dynamics and heat transfer problems. Scientific-Technical Bulletin of the St.-Petersburg State Technical University 2(36), 70–81 (2004). (in Russian)

9. Kozelkov, A.S., *et al.*: Multifunctional LOGOS software package for computing fluid dynamics and heat and mass transfer using multiprocessor computers: basic technologies and algorithms. Supercomputing and mathematical modeling: Proceedings of the XII International Workshop, Russia, Sarov, pp. 215–230 (2010). (in Russian)

10. Aksenov, A.A.: Flowvision: Industrial computational fluid dynamics. Computer Research and Modeling 9(1), 5–20 (2017). (in Russian) `https://doi.org/10.20537/2076-7633-2017-9-5-20`

11. Yao, Y., Yeo, K.-S.: An application of GPU acceleration in CFD simulation for insect flight. Supercomputing Frontiers and Innovations 4(2), 13–26 (2017). `https://doi.org/10.14529/jsfi170202`

12. Chaplygin, A.V., Gusev, A.V., Diansky, N.A.: High-performance shallow water model for use on massively parallel and heterogeneous computing systems. Supercomputing Frontiers and Innovations 8(4), 74–93 (2021). `https://doi.org/10.14529/jsfi210407`

13. Gorobets, A.V., Duben, A.P.: Technology for supercomputer simulation of turbulent flows in the good new days of exascale computing. Supercomputing Frontiers and Innovations 8(4), 4–10 (2022). `https://doi.org/10.14529/jsfi210401`

14. Shershnev, A.A., Kudryavtsev, A.N., Kashkovsky, A.V., Khotyanovsky, D.V.: HyCFS, a high-resolution shock capturing code for numerical simulation on hybrid computational clusters. AIP Conf. Proc. 1770, 030076 (2016). `https://doi.org/10.1063/1.4964018`

15. Kudryavtsev, A.N., Kashkovsky, A.V., Borisov, S.P., Shershnev, A.A.: A numerical code for the simulation of non-equilibrium chemically reacting flows on hybrid CPU-GPU clusters. AIP Conf. Proc. 1893, 030054 (2017). `https://doi.org/10.1063/1.5007512`

16. Borisov, S.P., Kudryavtsev, A.N., Shershnev, A.A.: Development and validation of the hybrid code for numerical simulation of detonations. J. Phys.: Conf. Ser. 1105, 012037 (2018). `https://doi.org/10.1088/1742-6596/1105/1/012037`

17. Borisov, S.P., Kudryavtsev, A.N., Shershnev, A.A.: Influence of detailed mechanisms of chemical kinetics on propagation and stability of detonation wave in H_2/O_2 mixture. J. Phys.: Conf. Ser. 1382, 012052 (2019). `https://doi.org/10.1088/1742-6596/1382/1/012052`

18. Alexander Burcat's Ideal Gas Thermodynamic Data in Polynomial form for Combustion and Air Pollution Use. `https://garfield.chem.elte.hu/Burcat/burcat.html`

19. McBride, B.J., Zehe, M.J., Gordon, S.: NASA Glenn coefficients for calculating thermodynamic properties of individual species. NASA/TP-2002-211556 (2002).

20. Gupta, R.N.: Viscous shock-layer study of thermochemical nonequilibrium. Journal of Thermophysics and Heat Transfer 10(2), 257–266 (1996). `https://doi.org/10.2514/3.801`

21. Landau, L., Teller, E.: Theory of sound dispersion. Phys. Z. Sowjetunion 10(1), 34–43 (1936).

22. Kustova, E., Oblapenko, G.: Reaction and internal energy relaxation rates in viscous thermochemically non-equilibrium gas flows. Phys. Fluids 27(1), 016102 (2015). `https://doi.org/10.1063/1.4906317`

23. Fedorov, A.V., Khmel, T.A. Fomin, V.M.: Non-equilibrium model of steady detonations in aluminum particles-oxygen suspensions. Shock Waves 9, 313–318 (1999). `https://doi.org/10.1007/s001930050191`

24. Fedorov, A.V., Khmel, T.A.: Numerical simulation of formation of cellular heterogeneous detonation of aluminum particles in oxygen. Combust. Expl. Shock Waves 41(4), 435–448 (2005). `https://doi.org/10.1007/s10573-005-0054-7`

25. Fedorov, A.V., Khmel, T.A.: Formation and degeneration of cellular detonation in bidisperse gas suspensions of aluminum particles. Combust. Expl. Shock Waves 44(3), 343–353 (2008). `https://doi.org/10.1007/s10573-008-0042-9`

26. Hirschfelder, J.O., Curtiss, C.F., Bird, R.B.: Molecular Theory of Gases and Liquids. Wiley, New York (1954).

27. Jiang, G.-S., Shu, C.-W.: Efficient implementation of weighted ENO schemes. J. Comput. Phys. 126, 202–228 (1996). `https://doi.org/10.1006/jcph.1996.0130`

28. Kolgan, V.P.: Application of the principle of minimal values of the derivative to construction of mesh schemes to calculation of discontinuous solutions of gas dynamics. Uch. Zap. TsAGI 3(6), 68–77 (1972). (in Russian). Translated to English and reprinted in J. Comput. Phys. 230, 2384–2390 (2011). `10.1016/j.jcp.2010.12.033`

29. van Leer, B.: Towards the ultimate conservative difference scheme. V. A second-order sequel to Godunov's method. J. Comput. Phys. 32(1), 101–136 (1979). `https://doi.org/10.1016/0021-9991(79)90145-1`

30. Anderson, W.K., Thomas, J.L., van Leer, B.: Comparison of finite volume flux vector splittings for the Euler equations. AIAA Journal 24(9), 1453–1460 (1986). `https://doi.org/10.2514/3.9465`

31. Einfeldt, B., Munz, C.D., Roe, P.L., Sjögren, B.: On Godunov-type methods near low densities. J. Comput. Phys. 92(2), 273–295 (1991). `https://doi.org/10.1016/0021-9991(91)90211-3`

32. Toro, E.F., Spruce, M., Speares, W.: Restoration of the contact surface in the Harten–Lax–van Leer Riemann solver. Shock Waves 4, 25–34 (1994). `https://doi.org/10.1007/BF01414629`

33. Batten, P., Leschziner, M.A., Goldberg, U.C.: Average-state Jacobians and implicit methods for compressible viscous and turbulent flows. J. Comput. Phys. 137, 38–78 (1997). `https://doi.org/10.1006/jcph.1997.5793`

34. Roe, P.L.: Approximate Riemann solvers, parameter vectors, and difference schemes. J. Comput. Phys. 43(2), 357–372 (1981). `https://doi.org/10.1016/0021-9991(81)90128-5`

35. van Leer, B.: Flux-vector splitting for the Euler equations. In: Krause, E. (eds) Eighth International Conference on Numerical Methods in Fluid Dynamics. Lecture Notes in Physics, vol. 170, pp. 507–512. Springer, Berlin, Heidelberg (1982). `https://doi.org/10.1007/3-540-11948-5_66`

36. Liou, M.-S., Steffen, C.J.: A new flux splitting scheme. J. Comput. Phys. 107(1), 23–39 (1993). `https://doi.org/10.1006/jcph.1993.1122`.

37. Edwards, J.R., Liou, M.-S.: Low-diffusion flux-splitting methods for flows at all speeds. AIAA J. 36, 1610–1617 (1998). `https://doi.org/10.2514/2.587`

38. Kim, K.H., Kim, C., Rho, O.-H.: Methods for the accurate computations of hypersonic flows – I. AUSMPW+ scheme. J. Comput. Phys. 174(1), 38–80 (2001). `https://doi.org/10.1006/jcph.2001.6873`

39. Liou, M.-S.: A sequel to AUSM, Part II: AUSM^{+}-up for all speeds. J. Comput. Phys. 214(1), 137–170 (2006). `https://doi.org/10.1016/j.jcp.2005.09.020`

40. Collins, J.P., Ferguson, R.E., Chien, K., *et al.*: Simulation of shock-induced dusty gas flows using various models. AIAA paper 94-2309.

41. Shu, C.-W., Osher, S.: Efficient implementation of essentially non-oscillatory shock-capturing schemes. J. Comput. Phys. 77(2), 439–471 (1988). `https://doi.org/10.1016/0021-9991(88)90177-5`

42. Thompson, R.J.: Improving round-off in Runge–Kutta computations with Gill's method. Commun. ACM 13(12), 739–740 (1970). `https://doi.org/10.1145/362814.362823`

43. Zhong, X.: Additive semi-implicit Runge–Kutta methods for computing high-speed nonequilibrium reactive flows. J. Comput. Phys. 128(1), 19–31 (1996). `https://doi.org/10.1006/jcph.1996.0193`

44. Wright, M.J., Candler, G.V., McDonald, J.D.: Data-parallel lower-upper relaxation method for reacting flows. AIAA Journal 32(12), 2380–2386 (1994). `https://doi.org/10.2514/3.12303`

45. Bondar, Ye.A., Markelov, G.N., Gimelshein, S.F., Ivanov, M.S.: Numerical modeling of near-continuum flow over a wedge with real gas effects. Journal of Thermophysics and Heat Transfer 20(4), 699–709 (2006). `https://doi.org/10.2514/1.18758`

46. Shuen, J.-S.: Upwind differencing and LU factorization for chemical non-equilibrium Navier–Stokes equations. J. Comput. Phys. 99(2), 233–250 (1992). `https://doi.org/10.1016/0021-9991(92)90205-D`

47. Hornung, H.G., Smith, G.H.: The influence of relaxation on shock detachment. J. Fluid Mech. 93(2), 225–239 (1979). `https://doi.org/10.1017/S0022112079001865`

DOI: 10.14529/jsfi220409

Numerical Study of Fuselage Impact on Acoustic Characteristics of a Helicopter Rotor[*]

Ilya V. Abalakin[1], *Vladimir G. Bobkov*[1], *Tatiana K. Kozubskaya*[1]

The paper presents the results of the simulation of unsteady turbulent flow generated by helicopter main rotor in the presence of the fuselage with an emphasis on the analysis of the influence of the fuselage on the rotor-induced flow and the rotor-generated acoustic field. The Reynolds-averaged Navier–Stokes equations with the Spalart–Allmaras turbulence model are used to simulate the Caradonna–Tung rotor and ROBIN fuselage interaction in hovering flight. The governing equations are discretized using the vertex-centered control volume method on mixed-element unstructured meshes with the sliding mesh technology to treat the rotor. The acoustic field generated by the rotor+fuselage interaction is comparatively analyzed against the case of an isolated rotor. It is found that the presence of fuselage significantly changes the rotor-generated acoustics. In particular, the presence of the fuselage noticeably distorts the directivity of acoustic radiation and increases the overall sound pressure level under the fuselage up to 20 dB, emphasizing the importance of the influence of fuselage on the helicopter acoustics.

Keywords: rotorcraft, CFD, rotor-fuselage interaction, RANS, sliding mesh, supercomputer.

Introduction

A growing number of publications [1–5] focusing on the influence of fuselage on the helicopter rotor aerodynamcis demonstrate the importance of the subject. Recently, due to increasingly stringent International Civil Aviation Organization (ICAO) requirements on the permissible noise levels produced by aircrafts, including helicopters, the focus of research has shifted from rotor aerodynamics to the acoustics generated by the rotor in the presence of fuselage [6–9]. The simulation of the acoustics of the entire helicopter including the fuselage presents a challenge as it requires substantial computational resources and efficient parallel algorithms.

Due to significant difference in the relative fuselage and blade velocities the aerodynamic noise generated by the flow around fuselage is negligible compared to the sound of helicopter rotor. However, the fuselage can strongly influence the rotor acoustics and its directivity diagram. This paper focuses on the numerical investigation of the influence of a fuselage on the tonal acoustics of the helicopter main rotor. For that purpose a higher-accuracy method for aeroacoustic simulations of rotor-fuselage interaction is developed.

The hovering helicopter flight is modelled by the Caradonna–Tung rotor [10] and ROBIN fuselage [11, 12], scaled to be compatible with the rotor radius. The choice of the helicopter model is motivated by the availability of detailed geometrical descriptions, experimental results, and prior experience simulating these separate cases [13].

When developing a method for numerical simulation of rotor-fuselage interaction the approach should be capable of modeling flows in the presence of both stationary and moving/rotating objects. The flow over an isolated helicopter rotor can be easily simulated in a non-inertial reference frame associated with the rotor [14, 15] using non-deformable fixed computational mesh. The simultaneous presence of moving (helicopter rotor) and stationary (fuselage) components makes it impossible to use a mono-block mesh. A standard solution is to

[*]One of the selected papers of The Ninth Russian conference "Computational Experiment in Aeroacoustics and Aerodynamics" (CEAA2022)

[1]Keldysh Institute of Applied Mathematics, Moscow, Russian Federation

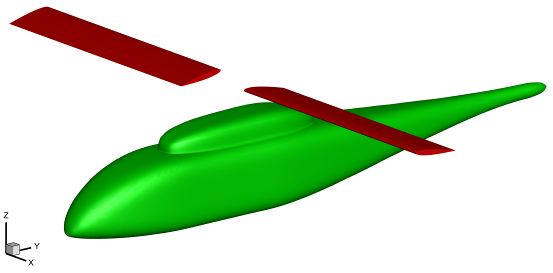
use sliding meshes, i.e., the decomposition of mesh into two or more subdomains with sliding non-overlapping interfaces one of which (the one containing the fuselage) being stationary in absolute reference frame while the rest of them rotating with rotors. In the present work, we exploit this approach placing the main rotor into the rotating puck-shape subdomain. Despite the fact that the approach is widely used [14, 16, 17], the application of vertex-centered higher-accuracy schemes brings additional difficulties in approximating the fluxes at the sliding interfaces. To handle this problem, an earlier developed technique presented in [18] is used.

In this work, turbulent flow over an isolated rotor and a rotor interacting with a fuselage in hovering flight including the near field acoustics for both cases is considered. The results are compared, and the influence of the fuselage on the acoustic field is evaluated.

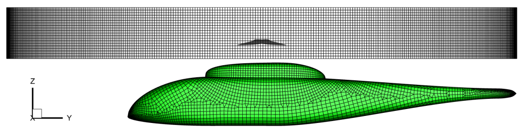
The paper is organized as follows. In Section 1, the problem of fuselage influence on the rotor tonal acoustics in hovering mode is formulated. The geometry of both components (i.e., the rotor and fuselage) are described, all the physical parameters are given. The governing mathematical model, the numerical methods in use and the computational set-up are presented in Sections 2, 3 and 4 respectively. Section 5 is devoted to the discussion of the obtained numerical results and to their comparative analysis. The concluding remarks and the acknowledgements are given at the end of the paper.

1. Problem Formulation

Turbulent flow over two-component configuration, consisting of two-bladed rotor and scaled fuselage model (Fig. 1) is considered. The rotor is modelled by the widely known Caradonna–Tung model [10], while the fuselage is approximated by the scaled version of ROBIN fuselage model [11, 12].



(a) Rotor and fuselage geometry



(b) Rotor and fuselage surface mesh

Figure 1. The case geometry and surface mesh

The rotor radius is $R = 1.143\ m$. The blades are based on the NACA 0012 airfoil and are untwisted and untapered. The blade chord is $0.1905\ m$ and the collective pitch angle is $8°$. A length of the ROBIN fuselage is scaled to $2R$. The distance between the rotor plane and the fuselage center of mass is $0.35\ m$. The gap between the fuselage pylon top and the rotor plane is about $0.125\ m$. The hovering flight is considered with the rotor rotation frequency $650\ RPM$, corresponding to the tip velocity $77.8\ m/s$. The Reynolds number defined in terms of rotor radius, tip velocity, the reference density $\rho_0 = 1.2046\ kg/m^3$, and molecular viscosity $\mu_0 = 1.8148 \times 10^{-5}\ Pa \cdot s$ is $Re = 5.9 \times 10^6$.

In addition to the aerodynamic characteristics, the pressure pulsation is measured in three sets of probes, namely the probes azimuthally distributed with angular step $10°$ in three planes including the rotor rotation plane (Fig. 2). The distance from each point to the rotor center is $2R$.

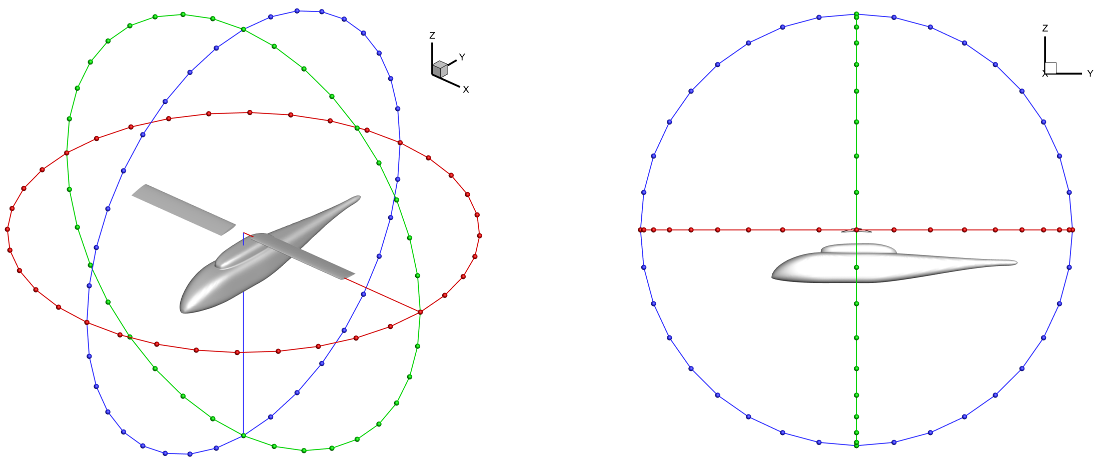


Figure 2. The sets of probes for acoustic field measurements: red – XY plane, green – XZ plane, blue – YZ plane

2. Governing Equations

The turbulent flow over the helicopter rotor is modelled by the unsteady Reynolds-averaged Navier–Stokes (RANS) equations with the closure model following the Boussinesq hypothesis. To simulate the turbulent flow near the rotor or rotor interacting with fuselage, the computational domain is decomposed in two subdomains, namely, a fixed subdomain considered in the absolute frame of reference (AFR) and a subdomain in the rotating frame of reference (RFR) that includes the rotor. The rotation velocity of RFR is taken equal to the angular velocity of rotor such that the rotor stationary in this frame of reference. In the fixed subdomain, the flow is governed by the RANS equations written in the AFR. To simulate the flow over the rotor, the unsteady RANS equations written in the RFR with the angular velocity $\boldsymbol{\omega}$ in terms of velocity vector defined in the AFR are solved. Both systems are closed by the Spalart–Allmaras turbulence model [19] governing the evolution of effective coefficient of turbulent viscosity. The unsteady RANS equations both in the AFR with $\boldsymbol{\omega} = 0$ and the RFR with $\boldsymbol{\omega} \neq 0$ have the following form:

$$\begin{aligned}
&\frac{\partial \rho}{\partial t} + \operatorname{div} \rho\left(\mathbf{u} - \mathbf{V}\right) = 0,\\
&\frac{\partial \rho \mathbf{u}}{\partial t} + \operatorname{Div} \rho\left(\mathbf{u} - \mathbf{V}\right) \bigotimes \mathbf{u} + \nabla p = \operatorname{Div} \mathbf{P} - \rho\left(\boldsymbol{\omega} \times \mathbf{u}\right),\\
&\frac{\partial \rho E}{\partial t} + \operatorname{div} \rho\left(\mathbf{u} - \mathbf{V}\right) E + \operatorname{div} \mathbf{u}p = \operatorname{div} \mathbf{q} + \operatorname{div} \mathbf{P}\mathbf{u},\\
&\frac{\partial \rho \widetilde{\nu}}{\partial t} + \operatorname{div} \rho \widetilde{\nu} \mathbf{u} = D_\nu + G_\nu - Y_\nu,
\end{aligned} \tag{1}$$

where $\mathbf{u}$ is the velocity vector, ρ is the density, $E = \frac{\mathbf{u}^2}{2} + \varepsilon$ is the total energy, ε is the specific internal energy, p is the pressure defined by ideal gas equation $p = (\gamma - 1)\,\rho\varepsilon$, $\gamma = 1.4$ is the specific ratio, d is the distance to the solid wall. In the system (1) $\mathbf{V}$ denotes the linear tangential velocity defined as $\boldsymbol{\omega} \times \mathbf{r}$ where $\mathbf{r}$ is the radius-vector. $\mathbf{P}$ is the viscous stress tensor defined by the strain rate tensor $\mathbf{S}$ according to the Boussinesq hypothesis in the following way:

$$P_{ij} = 2\mu_{eff}\left(S_{ij} - \frac{1}{3}\frac{\partial u_i}{\partial x_i}\delta_{ij}\right), \quad S_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right),$$

where the effective viscosity coefficient μ_{eff} equals to the sum of the coefficient of molecular viscosity μ and the turbulent eddy viscosity μ_T. Vector $\mathbf{q}$ defines the internal energy flux as[2]

$$q_i = \frac{\mu_{eff}}{\gamma \Pr} \frac{\partial \varepsilon}{\partial x_i},$$

where $\Pr = \frac{\mu_{eff} C_p}{\lambda}$ is the Prandtl number.

The coefficient of molecular viscosity μ is prescribed by the Sutherlands law

$$\mu = \mu_0 \frac{T_0 + S}{T + S} \left(\frac{T}{T_0} \right)^{\frac{3}{2}}, \ S = 120^\circ K,$$

where μ_0 and T_0 are the molecular viscosity and the free stream temperature.

The last equations of system (1) describe the evolution of the variable $\widetilde{\nu}$ used to calculate turbulent eddy viscosity $\mu_T = \rho f_{\nu 1} \widetilde{\nu}$. A specific form of the terms D_ν, G_ν and Y_ν describing the diffusion, turbulence generation and turbulence destruction as well as the definitions of the damping function $f_{\nu 1}$ and corresponding constants can be found, for instance, in [19].

For further considerations, it should be noted that, from the standpoint of an observer in the stationary frame of reference, the system of equations (1) describes the evolution of conservative variables due to their transport in the rotating (with velocity $\mathbf{V}$) media, the pressure gradient and the velocity vector turn to the angle equal to $\boldsymbol{\omega} t$ (implemented by the term $- \rho \ (\boldsymbol{\omega} \times \mathbf{u})$ in the momentum equation). Note that in the numerical implementation of this system, the rotation velocity can be interpreted as the velocity of moving mesh.

To simplify further approximation within finite-volume approach, the system of Reynolds-averaged Navier–Stokes equations (1) can be written in the following vector form with respect to the vector of conservative variables $\mathbf{Q} = (\rho, \rho \mathbf{u}, E, \rho \widetilde{\nu})^T$:

$$\frac{\partial \mathbf{Q}}{\partial t} + \nabla \cdot \left(\mathcal{F}^C (\mathbf{Q}) - \mathcal{F}^R (\mathbf{Q}) - \mathcal{F}^D (\mathbf{Q}, \nabla \mathbf{Q}) \right) = \mathbf{S} (\mathbf{Q}, \nabla \mathbf{Q}). \tag{2}$$

System (2) includes the composite vectors $\mathcal{F}^C$, $\mathcal{F}^R$ and $\mathcal{F}^D$, each component of which $\mathbf{F}_i^C$, $\mathbf{F}_i^R$ and $\mathbf{F}_i^D$ in coordinate direction x_i $(i = 1, 2, 3)$ represents the convective transport, rotation transport and diffusion flux vectors, respectively. Operator $(\nabla \cdot)$ is the divergence operator.

The convective transport, rotation transport and diffusion flux vectors are given as a function of the physical variables ρ, $\mathbf{u}$, p, $\widetilde{\nu}$ and their gradients

$$\mathbf{F}_i^C (\mathbf{Q}) = \left(\rho u_i, \rho u_i \mathbf{u} + p \mathbf{I}, (E + p) u_i, \rho \widetilde{\nu} u_i \right)^T,$$
$$\mathbf{F}_i^R (\mathbf{Q}) = \left(\rho V_i, \rho u_i \mathbf{V}, E V_i, \rho \widetilde{\nu} V_i \right)^T,$$
$$\mathbf{F}_i^D (\mathbf{Q}, \nabla \mathbf{Q}) = \left(0, P_{i1}, P_{i2}, P_{i3}, P_{ij} u_j + q_i, \frac{3}{2} (\mu + \rho \widetilde{\nu}) \frac{\partial \widetilde{\nu}}{\partial x_i} \right)^T,$$

where $\mathbf{I}$ – is the identity matrix.

Vector $\mathbf{S} (\mathbf{Q}, \nabla \mathbf{Q})$ is a source term describing the influence of the external forces that are not related to the transfer processes of the target variables $\mathbf{Q}$:

$$\mathbf{S} (\mathbf{Q}, \nabla \mathbf{Q}) = \left(0, \rho (\boldsymbol{\omega} \times \mathbf{u}), 0, G_\nu (\mathbf{Q}, \nabla \mathbf{Q}) - Y_\nu (\mathbf{Q}, \nabla \mathbf{Q}) + \frac{c_{b2}}{\sigma} \nabla \widetilde{\nu} \cdot \nabla \widetilde{\nu} \right)^T.$$

[2]We consider polytropic gas, so the specific heat c_V does not depend on temperature and, consequently, the internal energy is a linear function on temperature, i.e., $\varepsilon = c_V T$.

Here $c_{b2} = 0.622$ and $\sigma = 2/3$ according to the [19]. Note that the definition of diffusion term D_ν of the Spalart–Allmaras equation uses the square of the gradient of variable $\widetilde{\nu}$ which is of non-divergent form. Due to the last detail, we include this term to the source in the conservation law (2).

3. Numercial Method

The space discretization of the Navier–Stokes equations is based on the vertex-centered formulation implying that all the variables are defined at the vertices of hybrid unstructured mesh. The mesh vertices are the centers of the dual mesh cells. The approximation of the convective fluxes of the Navier–Stokes equations is built within the finite-volume approach. A cell of the dual mesh serves as control volume for which the discrete conservation laws are written. Thus, each control volume contains only one mesh vertex and belongs to the agglomeration of mesh elements containing this vertex.

Note that the mesh in each subdomain in corresponding reference frame is considered stationary, i.e., a geometry of each mesh element and respective control volume are fixed in time. Since the rotating subdomain is described in the RFR, the mesh there is also fixed and rotates with the rotor as a single entity.

The Navier–Stokes equations (2) are approximated using the integral form of these equations. The integral form of the system (2) for the control volume (or computational cell) K_i associated with the vertex i in the vertex-centered formulation can be written in the following form:

$$\int_{K_i} \frac{d\mathbf{Q}}{dt} dV + \int_{\partial K_i} \left(\mathcal{F}^C(\mathbf{Q}) \cdot \mathbf{n} - (\mathbf{V} \cdot \mathbf{n})\,\mathbf{Q}\right) dS = \int_{\partial K_i} \nabla \cdot \mathcal{F}^D(\mathbf{Q}, \nabla\mathbf{Q})\, dV + \int_{\partial K_i} \mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q})\, dV, \tag{3}$$

where ∂K_i is the boundary of control volume (or cell) K_i, $\mathbf{n}$ is the unit external normal to the boundary ∂K_i.

Let $|K_i|$ be the volume of computational cell, $\partial K_{ij} = K_i \cap K_j$ is the common part of boundaries of cells K_i and K_j (or segment), $N_1(i)$ is a set of nodes neighboring to the vertex i, $\overline{\mathbf{Q}}_i$ is the integral average of variable $\mathbf{Q}$ on the cell K_i. The approximation of the convective part of the Navier–Stokes equations (3) can be considered as an approximation of the Euler equations written in a form of conservation laws:

$$\frac{d\overline{\mathbf{Q}}_i}{dt} = -\frac{1}{|K_i|} \int_{\partial K_i} \left(\mathcal{F}^C(\mathbf{Q}) \cdot \mathbf{n} - (\mathbf{V} \cdot \mathbf{n})\,\mathbf{Q}\right) dS = \sum_{j \in N_1(i)} \int_{\partial K_{ij}} \left(\mathcal{F}^C \cdot \mathbf{n} - (\mathbf{V} \cdot \mathbf{n})\,\mathbf{Q}\right) ds$$
$$\approx \sum_{j \in N_1(i)} \mathbf{h}_{ij}\left(\mathbf{Q}^L_{ij}, \mathbf{Q}^R_{ij}\right) |\mathbf{n}_{ij}|,$$

where $\mathbf{h}_{ij}\left(\mathbf{Q}^L_{ij}, \mathbf{Q}^R_{ij}\right)$ is the numerical flux approximating the flux through the square of segment ∂K_{ij} on the base of Godunov-type schemes, and $|\mathbf{n}_{ij}|$ is the absolute value of oriented square of segment

$$\mathbf{n}_{ij} = \int_{\partial K_{ij}} \mathbf{n}\, dS. \tag{4}$$

The normal velocity of rotation (or, the velocity of moving mesh) is approximated an average on the segment as

$$V_{ij} = \frac{\mathbf{1}}{|\mathbf{n_{ij}}|} \int_{\partial K_{ij}} \mathbf{V} \cdot \mathbf{n} \, dS. \tag{5}$$

The value of variables $\mathbf{Q}_{ij}^{L/R}$ taken from the left and right from segment ∂K_{ij} are determined by the quasi-one-dimensional edge-based reconstruction on the extended stencil [20–22] that results in increasing the accuracy order of approximation of convective fluxes. The extended stencil contains $2n + 1$ points, and not all of them coincide with the mesh nodes. In the last case, the values in such points are found by the linear interpolation on the corresponding faces of mesh elements which have the intersection with the straight edge ij – containing line.

The approximation of viscous fluxes given by the volume integral $\int_{\partial K_i} \nabla \cdot \mathcal{F}^D(\mathbf{Q}, \nabla\mathbf{Q})\, dV$ in the Navier–Stokes equation (3) is implemented by the Galerkin method basing on the linear polynomials. The integral of source term $\mathbf{S}$ in the system (3) is approximated as a volume average as

$$\int_{K_i} \mathbf{S}(\mathbf{Q}, \nabla\mathbf{Q})\, dV \approx |K_i|\, \mathbf{S}(\mathbf{Q}_i, (\nabla\mathbf{Q})_i),$$

where the derivatives of variables $(\nabla\mathbf{Q})_i$ including to the source $\mathbf{S}$ are approximated by the discrete nodal gradients defined as a weighted sum of gradients on the mesh elements having the common vertex in node i [23].

The time integration is carried out using the implicit three-layer scheme of the second approximation order followed by the Newton linearization of the space-discretized equations. At each Newton iteration to solve the system of linear equations we use the stabilized method of biconjugate iterations (BiCGSTAB) [24]. The details of this approach and its usage in simulations are discussed in [25].

The presence of the stationary fuselage and the rotating blades results in the necessity to use two mesh blocks containing stationary and moving objects correspondingly and, consequently, to provide a proper interaction between these blocks.

A standard solution in this situation is the usage of sliding meshes, i.e., the decomposition of the mesh in two or more subdomains, one of which is at rest in the AFR while the rest of them rotate with the rotors they contain.

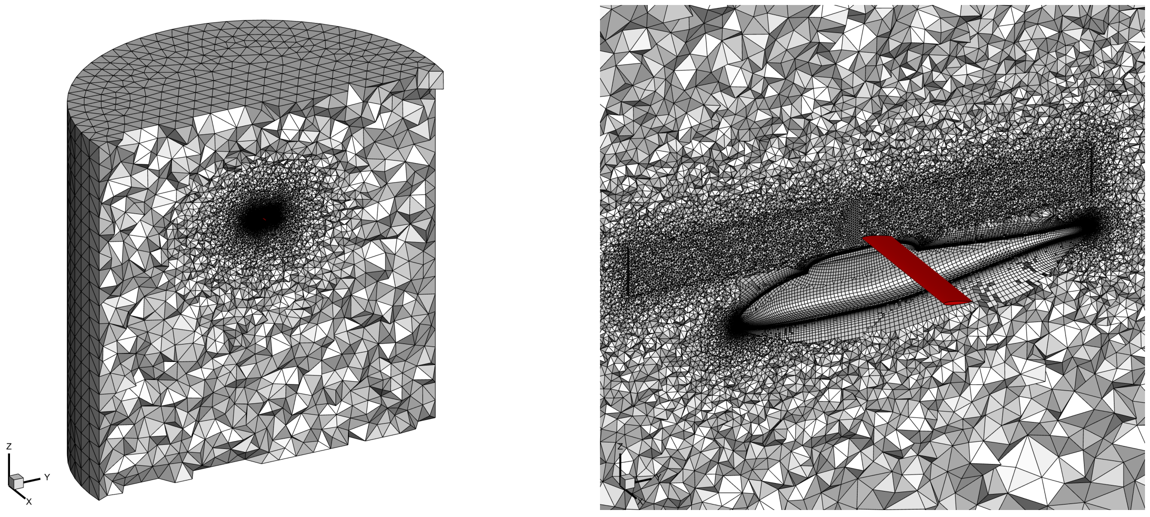
For the element-centered finite-volume methods assuming the variable averaging over unstructured mesh elements, the formulation providing the mesh sliding can be written without significant loss of accuracy. For the vertex-centered schemes assuming the variable averaging over dual cells, it is much harder to develop a method without any loss of accuracy on the interface. For the vertex-centered schemes for unstructured meshes in the general case, there is no known solution capable to preserve the high accuracy on sliding meshes. Nevertheless, a loss of accuracy can be compensated by a better scheme resolution and local mesh refinement near sliding interfaces. The following two versions of sliding mesh interfaces [18] have been developed:

- interface separating fixed and rotating mesh blocks is plane (for instance, a circle, an annual segment or a plane);
- interface separating fixed and rotating mesh blocks is a lateral surface of a finite straight round cylinder.

As a result, the rotating domain can be a finite straight round cylinder that includes both possible types of interface, or, in other words, can have a puck-shaped configuration. Within the developed approach, the control volumes (dual cells) are built in a way that the agglomerations of cells in the fixed and rotation subdomains do not overlap and have the curvilinear sliding surface as the only common part. The cell faces belong to the lateral surface and thereby have its curvilinear shape. After the dual cells are defined, we calculate the values of $\mathbf{n}_{ij}$ and V_{ij} following the formula (4) and (5) with account for possible curvilinear shape of the cell faces.

4. Computational Set-up

As it is mentioned above, two types of simulations are performed, namely, the simulation of the isolated rotor in the cylindrical rotating region with the sliding interface (rotor simulation (RS)) and the simulation of the same rotor supplemented by the fuselage (rotor+fuselage simulation (RFS)).



(a) overall view (b) fuselage mesh

Figure 3. The unstructured mesh used in RFS case

For all the simulations, we build the mixed-element unstructured mesh consisting of tetrahedrons, prisms, pyramids and hexahedrons. In all the cases, the rotor surface mesh and the boundary layer prismatic mesh is the same, and the first surface cell height is chosen to meet the $y^+ < 1$ criteria. The sliding meshes represent two topologically separated mesh blocks: the outer fixed unstructured mesh block (Fig. 3a) and the inner rotating unstructured mesh block containing the rotor (Fig. 3b). The rotating puck-shaped domain is a cylinder with radius 1.5 m and height 0.3 m. The maximum size of the mesh element inside the rotating region is limited by 15 mm. The computational domain in both simulations is a cylinder with radius 30 m and height 60 m. The mesh sizes for both cases are represented in the Tab. 1.

Table 1. Computational mesh sizes

Case	Number of nodes	Number of cells
RS	6 820 497	21 360 814
RFS	7 156 503	21 773 730

The height of the mesh elements in the location of acoustic probes (see Sec. 1) is 8–10 cm that well resolves the BPF frequencies.

5. Numerical Results

5.1. Aerodynamic Characteristics of Isolated Rotor

The aerodynamic characteristics of Caradonna–Tung rotor in hover simulated in the RFR were predicted in our previous work [13]. The results showed a good agreement with the experimental data for pressure coefficient distribution and the tip vortex trajectory. Here we validate the AFR with sliding mesh approach on the same cases and obtain practically the same good results with respect to the available experimental data. Figure 4 demonstrates the pressure coefficient distribution in a set of the rotor blade sections.

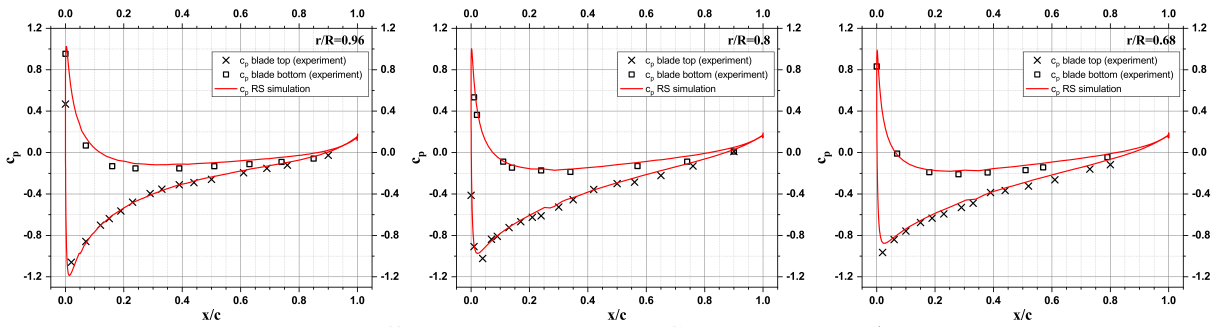
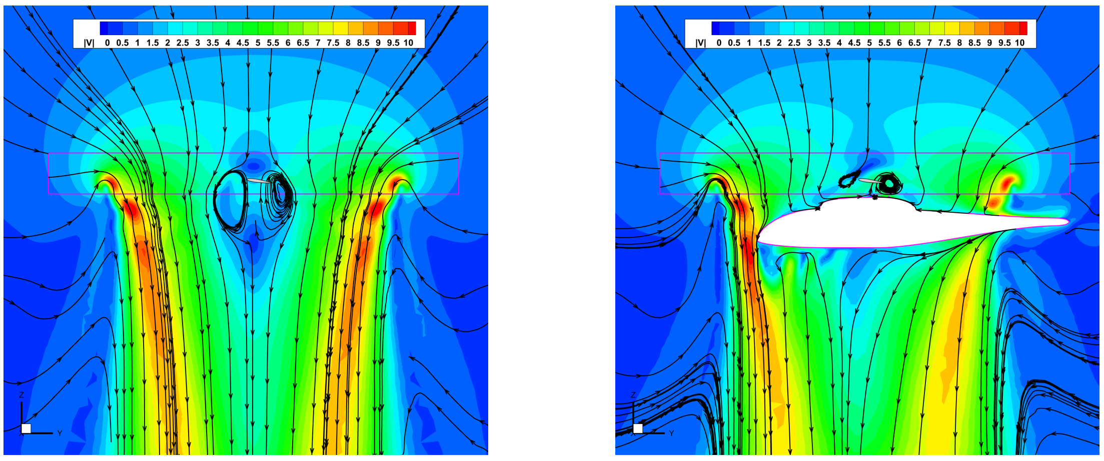


Figure 4. The pressure coefficient distribution: RS simulation in AFR vs. the experiment

5.2. Simulation of Turbulent Flow near Rotor+Fuselage Configuration

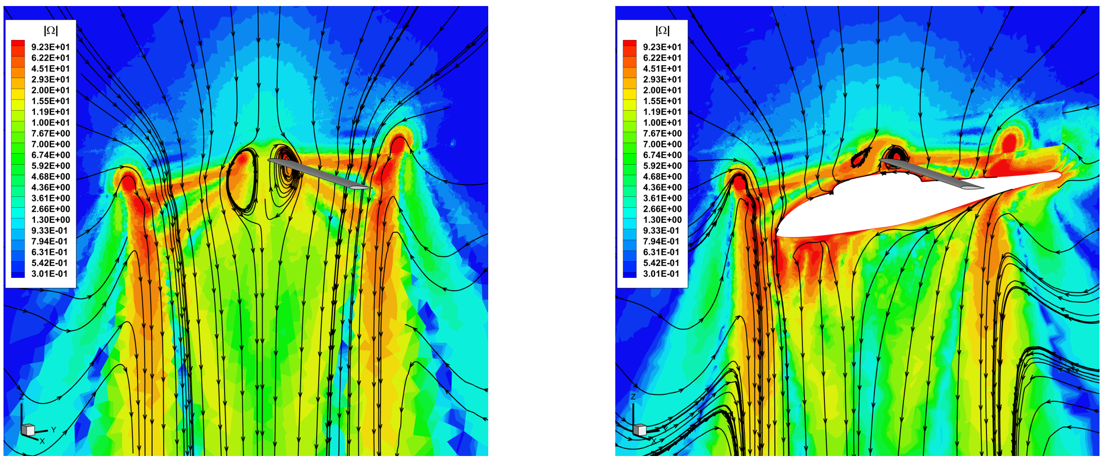
The flow field obtained in the RS presents a typical axial flow near a helicopter rotor (Fig. 5a–7a). The figures show the flow field and the wake behind the hovering rotor. The flow downstream the rotor is symmetrical as expected for the hovering mode. Besides, one can notice the slow vortex near the blades roots. The maxima of the velocity magnitude corresponds to the region of tip vortices with the downstream shift to the rotation axis.


|V| 0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6 6.5 7 7.5 8 8.5 9 9.5 10
Z
Y

(a) RS case (b) RFS case

Figure 5. The velocity magnitude field with streamlines in YZ section

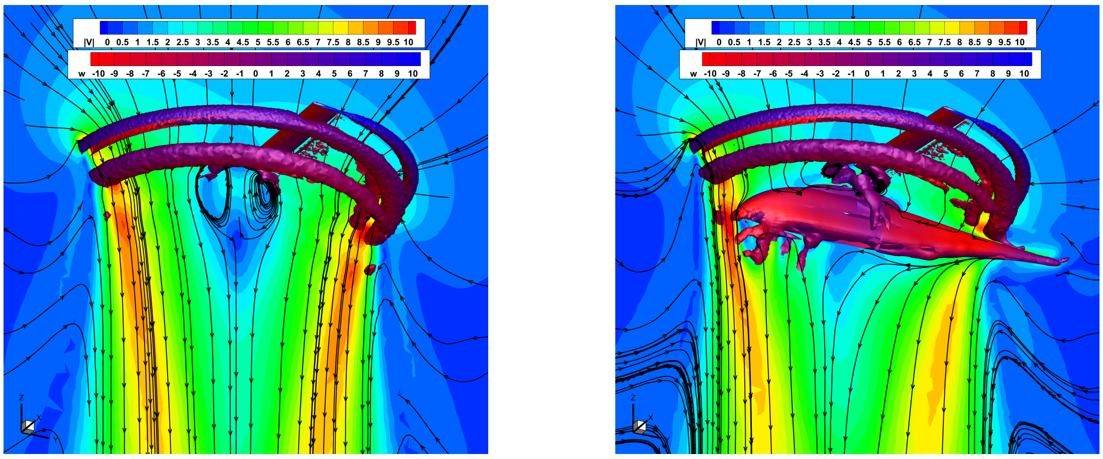
Figure 5b shows that presence of the fuselage strongly affects the flow structure downstream the rotor. The flow becomes asymmetrical, and the most significant impact is the induced flow slowdown under the fuselage. The vorticity field indicates the intensive vortex shedding downstream the fuselage body and the effect of interaction between the rotor tip vortices with the fuselage nose and its tail boom (see Fig. 6b, 7b).



(a) RS case (b) RFS case

Figure 6. The vorticity magnitude field with streamlines in YZ section

As seen Fig. 7 with the Q-criterion visualization, both RS and RFS capture two revolutions of the wake spiral downstream the rotor.



(a) RS case (b) RFS case

Figure 7. The velocity magnitude field with streamlines in YZ section and the Q-criterion isosurface colored in the vertical flow velocity

5.3. Impact of Fuselage on Rotor Acoustics

The acoustic characteristics estimated basing on the simulation results are the spectra of pressure pulsation in the probes, and the directivity diagrams of blade passing frequency (BPF) sound pressure level (SPL) and overall sound pressure level (OASPL). The OASPL is determined by the pressure pulsations spectrum $P(\mathbf{x}, f)$ in the particular probes. The pressure pulsation spectrum is given by the Fourier transform of the function $p(\mathbf{x}, t)$ normalized by the quantity $p_0 = 2 \times 10^{-5}\ Pa$:

$$P(\mathbf{x}, f) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \frac{p(\mathbf{x}, t)}{p_0} e^{-i2\pi f} dt.$$

The total energy of the spectrum is calculated as the integral of the spectral power $S(\mathbf{x}, f) = P^2(\mathbf{x}, f)$: as

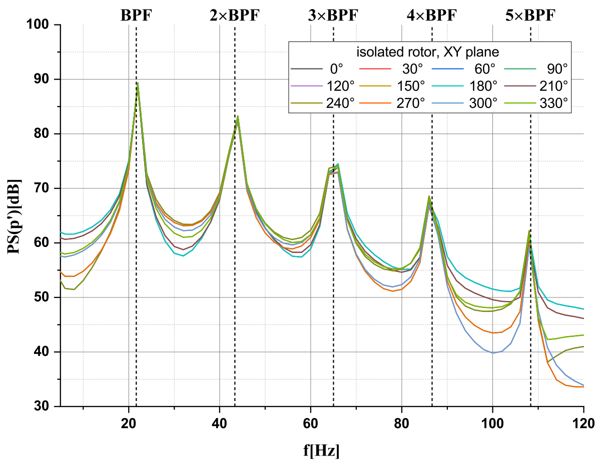
$$E(\mathbf{x}) = \int S(\mathbf{x}, f)\, dt,$$

where the integration is performed over all the resolvable frequencies. The OASPL measured in decibels is found by the formula

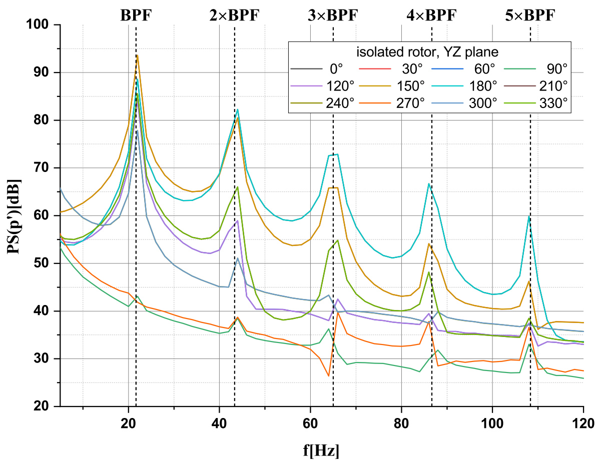
$$OASPL = 10 \log_{10} E(\mathbf{x}).$$

All the spectra presented are built using the fast Fourier transform on the probes pressure signals with 2 Hz sampling frequency.

It should be noted that RANS method used in paper is capable to capture only the tonal acoustics, i.e., the acoustic harmonics at BPF and its multiples. In the setup for two-bladed rotor at 650 RPM, the BPF value is 21.6 Hz. The obtained spectrum of pressure pulsation in the probes confirms that the peak tonal frequencies are reached at the multiples of BPF (see Fig. 8). As expected, in plane XY of the rotor rotation the amplitudes of pressure pulsation in all the probes at BPF and its multiples have the same values (see Fig. 8a and Fig. 9a).



(a) XY plane

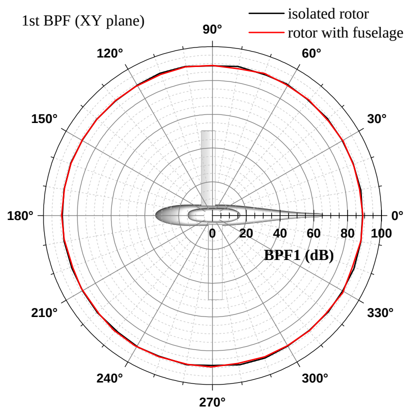


(b) YZ plane

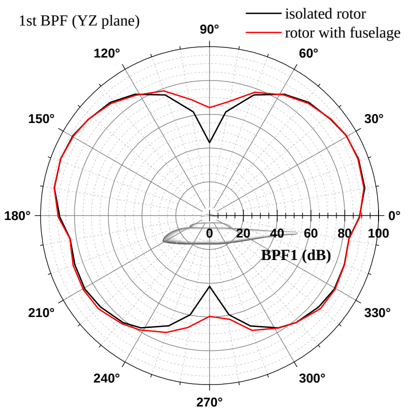
Figure 8. The pressure pulsation spectra measured in the probes obtained in RS

Figure 9 shows the directivity diagram of first BPF SPL in absence and presence of the fuselage at three orthogonal planes. The fuselage impact on the near-field acoustics is clearly seen. It should be noted that for the isolated rotor in hovering mode the pressure pulsation measured at the rotation axes must be zero theoretically. In our case, it is not so (it is about 40–45 dB) solely because of the asymmetric computational setup. This fact is confirmed by our simulation of a single blade with the periodicity condition in azimuth where it is really zero (see [15]). The presence of the fuselage makes the configuration asymmetrical and it results in a slight asymmetry in the directivity diagram in planes XZ and YZ near the rotation axes. A noticeable SPL increase in comparison with the case of isolated rotor is also detected there. One can see up to $\sim 7\ dB$ amplitude growth at the $\pm 20^\circ$ direction from the rotation axis in YZ plane and $\sim 7\ dB$ increase at azimuth $260^\circ < \varphi < 290^\circ$ in XZ plane. The difference in the directivity diagrams of first BPF SPL in the cases of isolated rotor and rotor with the fuselage is insignificant for the rest of azimuths.

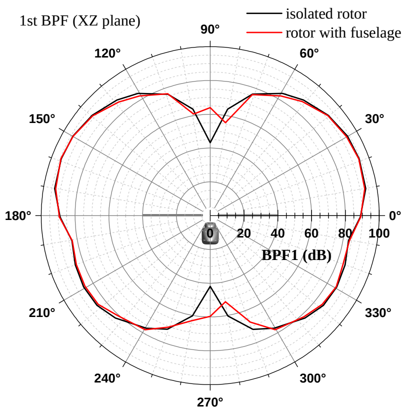
Figure 10 demonstrates the directivity diagram of OASPL in absence and in presence of the fuselage in different planes. The asymmetry of the OASPL diagram in XZ and YZ planes due to



(a) XY plane



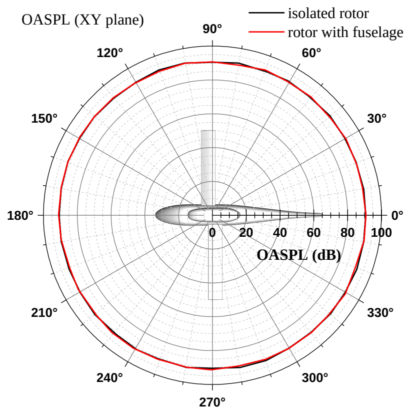
(b) YZ plane



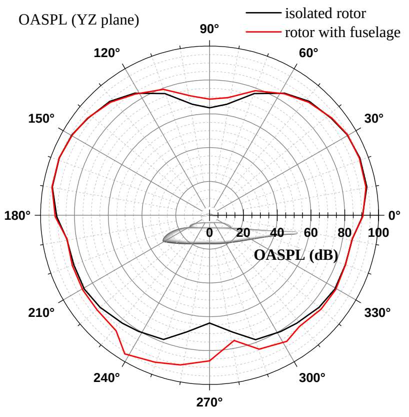
(c) XZ plane

Figure 9. The directivity diagram of first BPF SPL

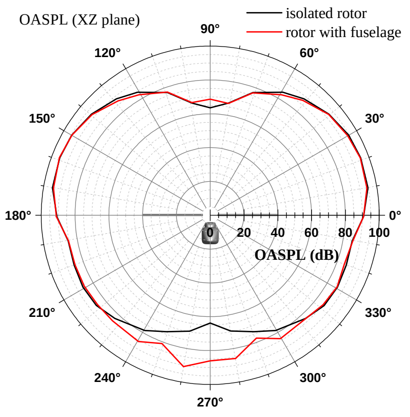
the fuselage presence becomes stronger in comparison with BPF SPL measurements. The OASPL near the rotation axes increases even more which can indicate a greater contribution of multiple frequencies. In YZ plane, the maximum difference observed on the azimuth $220° < \varphi < 300°$ and it reaches $\sim 20\ dB$. In the XZ, plane the maximum difference observed on the azimuth $240° < \varphi < 300°$ and it reaches $\sim 19\ dB$. Most likely, the reason of this OASPL symmetric growth is the blade interaction with the fuselage nose and tail boom at the azimuthal blade positions near 180° and 0° respectively (see Fig. 10a) that results in strengthening of scattering effects and, consequently, of acoustic radiation at multiple frequencies. In particular, it is seen that the azimuthal size of deformation of the directivity diagram is directly linked with the azimuthal size of the fuselage (see Fig. 10b, 10c).



(a) XY plane



(b) YZ plane



(c) XZ plane

Figure 10. The OASPL directivity diagram

Conclusion

The paper presents the results of numerical simulation of tonal noise generated by the helicopter rotor and, which is more interesting, evaluates the impact of the fuselage on the generated acoustic field. It is shown that the presence of fuselage may significantly strengthen the acoustic radiation in the direction towards the ground and noticeably deform its directivity diagram. In particular, the most significant influence of the fuselage is detected at the probes in the low half of the diagram measured at two radiuses from the rotor center in vertical planes where the OASPL growth reaches 20 *dB*. The discovered phenomenon of the possible amplification of acoustic radiation towards the ground due to the presence of the fuselage is of great importance for controlling the sound level generated by the helicopter. This result undoubtedly requires further study, and its thorough verification and validation.

The considered configuration "main rotor + fuselage" is an example of problems that are difficult to study in detail in wind tunnels. Field experiments related to the determination of the acoustic properties of entire helicopters are expensive and associated with the difficulties of accurate measurements in real operating conditions. In this situation, computational experiments may be in high demand. However, a natural obstacle on this way is the large computational resources required to compute such problems. This difficulty can be overcome or significantly mitigated by using modern high-performance systems with high efficiency, provided, for example, as in this work, by the efficient heterogeneous CPU + GPU parallelization model. The benefit of using HPC systems is further enhanced by the usage of higher-accuracy numerical methods and well-tuned mathematical models to accurately predict unsteady turbulent flows along with the generated acoustic fields.

Acknowledgements

The research is carried out using the equipment of the shared research facilities of HPC computing resources at Lomonosov Moscow State University, computing resources of the federal collective usage center Complex for Simulation and Data Processing for Mega-science Facilities at NRC "Kurchatov Institute", `http://ckp.nrcki.ru` and the hybrid supercomputer K60 installed in the Supercomputer Centre of Collective Usage of KIAM RAS.

The work was funded by the Russian Science Foundation within Project No. 20-41-09018.

References

1. Park, Y.M., Kwon, O.J.: Simulation of Unsteady Rotor Flow Field Using Unstructured Adaptive Sliding Meshes. Journal of the American Helicopter Society 49(4), 391–400 (2004). `https://doi.org/10.4050/JAHS.49.391`

2. Xu, H.-Y., Xing, S.-L., Ye, Z.-Y., Ma, M.-S.: A simple and conservative unstructured sliding-mesh approach for rotor-fuselage aerodynamic interaction simulation. Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering 231(1), 163–179 (2017). `https://doi.org/10.1177/0954410016664919`

3. Steijl, R., Barakos, G.N.: Sliding mesh algorithm for CFD analysis of helicopter rotor-fuselage aerodynamics. Int. J. Numer. Meth. Fluids 58(5), 527–549 (2008). `https://doi.org/10.1002/fld.1757`

4. Steijl, R., Barakos, G.N.: Computational Study of Helicopter Rotor-Fuselage Aerodynamic Interactions. AIAA J. 47(9), 2143–2157 (2009). `https://doi.org/10.2514/1.41287`

5. Jung, M.-S., Kwon, O.-J.: Numerical Simulation of Unsteady Rotor Flow Using an Unstructured Overset Mesh Flow Solver. Int'l J. of Aeronautical, Space Sciences 10(1), 104–111 (2009). https://doi.org/10.5139/IJASS.2009.10.1.104

6. Goulos, I.: Modelling the aeroelastic response and flight dynamics of a hingeless rotor helicopter including the effects of rotor-fuselage aerodynamic interaction. The Aeronautical Journal 119(1214), 433–478 (2015). http://doi.org/10.1017/S0001924000010563

7. Cheng, Q., Zhu, Y., Feng, Z., Chen, Q.: A Coupled Helicopter Rotor/Fuselage Dynamics Model Using Finite Element Multi-body. MATEC Web Conf. 7, 01016 (2016). http://doi.org/10.1051/matecconf/20167701016

8. Kusyumov, A.N., Mikhailov, S.A., Garipova, L.I., *et al.*: Distribution of Acoustic Power Spectra for an Isolated Helicopter Fuselage. EPJ Web of Conferences 114, 02062 (2016). http://doi.org/10.1051/epjconf/201611402062

9. van der Wall, B.G., Yin, J.: A model for real-time computation of fuselage-rotor interference. International Journal of Modeling, Simulation, and Scientific Computing 08(04), 1743002 (2017). http://doi.org/10.1142/s1793962317430024

10. Caradonna, F.X., Tung, C.: Experimental and Analytical Studies of a Model Helicopter Rotor in Hover. Technical report, NASA Technical Memorandum TM-81232, 1981.

11. Freeman, C.E., Mineck, R.E.: Fuselage surface pressure measurements of a helicopter wind-tunnel model with a 3.15-meter diameter single rotor. Technical report, NASA Technical Memorandum TM-80051, 1979.

12. Hillier, B.B., Stock, M.J., Gharakhani, A. Robust and Corrected Coefficients for the ROtor-Body-INteraction Body. AIAA J. 59(10), 4281–4283 (2021). http://doi.org/10.2514/1.J060303

13. Bobkov, V. Abalikin, I., Kozubskaya, T.: Simulation of Helicopter Rotors on Unstructured Mixed Meshes Using Edge-Based Reconstruction Schemes. In 14th WCCM-ECCOMAS Congress, CIMNE (2021). http://doi.org/10.23967/wccm-eccomas.2020.308

14. Barakos, G., Vahdati, M., Sayma, A.I., *et al.*: A Fully Distributed Unstructured Navier–Stokes Solver for Large-Scale Aeroelasticity Computations, The Aeronautical Journal 105(1050), 419–426 (2016). http://doi.org/10.1017/S0001924000012392

15. Abalakin, I.V., Anikin, V.A., Bakhvalov, P.A., *et al.*: Numerical Investigation of the Aerodynamic and Acoustical Properties of a Shrouded Rotor. Fluid Dyn. 51(3), 419–433 (2016). http://doi.org/10.1134/S0015462816030145

16. Bachler, G., Schiffermller, H., Bregant, A.: A Parallel Fully Implicit Sliding Mesh Method for Industrial CFD Applications. Parallel Computational Fluid Dynamics 2000, 501–508 (2001). http://doi.org/10.1016/B978-044450673-3/50129-9

17. Titarev, V.A., Faranosov, G.A., Chernyshev, S.A., Batrakov, A.S.: Numerical Modeling of the Influence of the Relative Positions of a Propeller and Pylon on Turboprop Aircraft Noise. Acoust. Phys. 64, 760–773 (2018). http://doi.org/10.1134/S1063771018060118

18. Abalakin, I.V., Bakhvalov, P.A., Bobkov, V.G., Gorobets, A.V.: Parallel Algorithm for Flow Simulation in Rotor-Stator Systems Based on Edge-Based Schemes. Math. Models Comput. Simul. 13(1), 172–180 (2021). http://doi.org/10.1134/S2070048221010026

19. Spalart, P.R., Allmaras, S.R.: A One-Equation Turbulence Model for Aerodynamic Flows. 30th Aerospace Sciences Meeting, Exhibit, Reno, NV, USA, January 6–9, 1992. AIAA Paper 92-0439. http://doi.org/10.2514/6.1992-439

20. Abalakin, I., Bakhvalov, P., Kozubskaya, T.: Edge-based reconstruction schemes for unstructured tetrahedral meshes. Int. J. Numer. Meth. Fluids 81(6), 331–356 (2016). http://doi.org/10.1002/fld.4187

21. Bakhvalov, P.A., Kozubskaya, T.K.: Construction of edge-based 1-exact schemes for solving the Euler equations on hybrid unstructured meshes. Comput. Math. Math. Phys. 57(4), 680–697 (2017). http://doi.org/10.1134/S0965542517040030

22. Bakhvalov, P., Kozubskaya, T.: EBR-WENO scheme for solving gas dynamics problems with discontinuities on unstructured meshes. Comput. Fluids 157, 312–324 (2017). http://doi.org/10.1016/j.compfluid.2017.09.004

23. Dervieux, A., Alauzet, F., Loseille, A., Koobus, K.: Mesh Adaptation for Computational Fluid Dynamics 1: Continuous Riemannian Metrics and Feature-based Adaptation. ISTE-Wiley Publishing (2022). http://doi.org/10.1002/9781394163991

24. van der Vorst, H.A.: Bi-CGSTAB: A Fast and Smoothly Converging Variant of Bi-CG for the Solution of Nonsymmetric Linear Systems. SIAM J. Sci. Stat. Comput. 13(2), 631–644 (1992). http://doi.org/10.1137/0913035

25. Abalakin, I.V., Bobkov, V.G., Kozubskaya, T.K.: Implementation of the low Mach number method for calculating flows in the NOISEtte software package. Math. Models Comput. Simul. 9(6), 688–696 (2017). http://doi.org/10.1134/S2070048217060023

26. Gorobets, A., Bakhvalov, P.: Heterogeneous CPU+GPU parallelization for high-accuracy scale-resolving simulations of compressible turbulent flows on hybrid supercomputers. Computer Physics Communications 271, 108231 (2021). http://doi.org/10.1016/j.cpc.2021.108231

27. Gorobets, A.V., Duben, A.P.: Technology for Supercomputer Simulation of Turbulent Flows in the Good New Days of Exascale Computing. Supercomputing Frontiers and Innovations 8(4), 4–10 (2022). http://doi.org/10.14529/jsfi210401