



## Data compression for turbulence databases using spatiotemporal subsampling and local resimulation

Zhao Wu , Tamer A. Zaki, and Charles Meneveau 

*Department of Mechanical Engineering, Johns Hopkins University, Baltimore, Maryland 21218, USA*



(Received 26 October 2019; accepted 19 May 2020; published 15 June 2020)

Motivated by specific data and accuracy requirements for building numerical databases of turbulent flows, data compression using spatiotemporal subsampling and local resimulation is proposed. Numerical resimulation experiments for decaying isotropic turbulence based on subsampled data are undertaken. The results and error analyses are used to establish parameter choices for sufficiently accurate subsampling and subdomain resimulation.

DOI: [10.1103/PhysRevFluids.5.064607](https://doi.org/10.1103/PhysRevFluids.5.064607)

### I. INTRODUCTION

In the field of computational fluid dynamics, the study of turbulent flows based on data generated using direct numerical simulation (DNS) has occupied a prominent place in the literature over the past several decades [1–5].

DNS provides spatial and temporal resolution down to the smallest and fastest eddies of a turbulent flow. Therefore, the Reynolds number achievable by DNS is limited by computing power and memory, and has been growing roughly at the rate expected from Moore’s law. The amount of data generated by DNS is growing accordingly [6–10]. For instance, a simulation of turbulent flow outputting four field variables (e.g., the three velocity components and pressure) on  $2000^3$  spatial grid points and integrated over, say,  $5 \times 10^4$  time steps, will generate several Petabytes (PB) of data. Researchers thus store only a few selected snapshots of the flow during the simulations, and primarily rely on run-time analysis tools that are decided prior to the computation if time resolved phenomena are to be studied. As a result, when new questions and concepts arise, massive simulations must be performed over and over. Moreover, when storing snapshot data for later analysis, the traditional means of sharing available data after DNS, e.g., [11], assumes that the data are downloaded as flat files and consequently a user has to worry about formats and provide the computational resources for analysis.

As a means to address these problems that challenge further growth of DNS and accessibility of data, modern database technologies have begun to be applied to DNS-based turbulence research. For instance, the Johns Hopkins Turbulence Database (JHTDB) [12–14] has been constructed and has been in operation for about a decade, as an open public numerical laboratory. The system hosts about 1/2 PB of DNS data including five space-time resolved data sets and several others with a few snapshots available. Users have Web-service facilitated access to the data, among others using a “virtual sensors” approach in which a user specifies the position and time at which data are requested and the system returns properly interpolated field data. Other derived quantities such as gradients [14] and fluid trajectories [15] are also available, typically delivered to within single-precision machine accuracy. A hallmark of the system is the ability of users to access very small targeted subsets of the data without having to download the entirety of the data. The system has been successful at democratizing access to some of the world’s largest high-fidelity DNS of canonical turbulent flows. JHTDB data have been used in over 160 peer-reviewed journal articles since its inception, about 40 in 2019 alone.

In recent years, the scale of DNS data has continued to grow further. The largest simulations now generate data on about  $O(10^4)$  grid points in each of the three directions, so storing multiple time steps to capture time evolution becomes very challenging, even in efficiently built databases. For example, storing even only one large-scale turnover time of the  $8192^3$  isotropic turbulence data set [6] would require storing about 80 PB. Over the next several years, it can be anticipated that even larger-scale DNS will be performed, generating exabytes of data, far out of reach of anticipated facilities and the approaches on which JHTDB is currently based.

It is therefore necessary to explore innovative tools for compressing simulation data for use in conjunction with databases. Most of the general-purpose data compression algorithms are based on analyzing the data representation, and can generally be classified as lossless or lossy. Lossless data compression utilizes statistical redundancy [e.g. 16,17], while lossy data compression is to remove unnecessary data, e.g., JPEG [18] and MP3 [19]. Lossless data compression tools are promising, but for turbulence data where the flow's small-scale structures contain nontrivial information at each grid point the compression ratios can be expected to be somewhat limited. While we continue current efforts along this direction and can expect further improvements, more aggressive tools will be required for the very large data sets envisioned in the near future. Regarding lossy compression, it is certainly appropriate for visualization and other applications where less fidelity is acceptable. However, if one wishes, e.g., to capture accurate velocity gradients, lossy compression algorithms in which the accuracy of primary variables is degraded, say, at the fourth decimal point, will already lead to significant errors in gradients and will thus be insufficient for the purposes of turbulence research.

It bears recalling that JHTDB enables users to receive interpolated data between spatial and temporal grid points, using polynomial functions (Lagrange, spline, Hermite). Far more aggressive data compression could be achieved if data could be stored more sparsely in both space and time. However, when a user requests localized pieces of data that fall between coarsely stored positions and/or times, one would need to revert to the dynamical equations (i.e., Navier-Stokes) to perform a physics-based rather than a polynomial-based interpolation.

In this paper we explore and establish requirements for such a data compression method, named “spatiotemporal subsampling and subdomain resimulation” (STSR). The method aims at enabling users to recover data at close to machine accuracy (single-precision), based on very coarsely stored data. While the method can greatly compress the amount of data to be stored, such savings have to be balanced by the additional cost of processor (CPU or GPU) expense needed later on to accommodate user queries.

Initial efforts attempting to reproduce DNS data using local resimulation (technical details to be provided below) have shown a surprisingly narrow and stringent range of conditions under which resimulation in a subdomain can generate data at the desired accuracy. That is to say, resimulation that reproduces DNS at close to single-precision machine accuracy, the desired baseline accuracy level, is more difficult to achieve than one may expect. Any small deviations from the conditions to be developed can be shown to lead to significant errors. It will be observed that the errors do not arise due to chaotic dynamics as we do not observe exponential divergence of state-space trajectories or exponential growth of errors over time. The absence of chaotic divergence of dynamics may be due to the strong constraints introduced by boundary conditions prescribed around closed subdomains, i.e., that the ratio of subdomain size to viscous length scale is sufficiently small for synchronization of chaos [20,21] to occur in the cases tested. Instead, errors are introduced due to small details of numerical implementation, discretization, and order of operations that at first glance may appear small and trivial but that can cause rather significant differences in results.

Therefore, the present paper aims to document the technical methodologies and tests performed with considerable attention to detail. Section II introduces the basic idea of data compression for turbulence databases using STSR. The desire to enable resimulations over localized spatial domains precludes the use of spectral methods based on global basis functions. In this paper, we explore the use of one of the most common discretization tools in computational fluid dynamics (CFD): second-order finite differencing. The numerical scheme adopted in the present computations is described

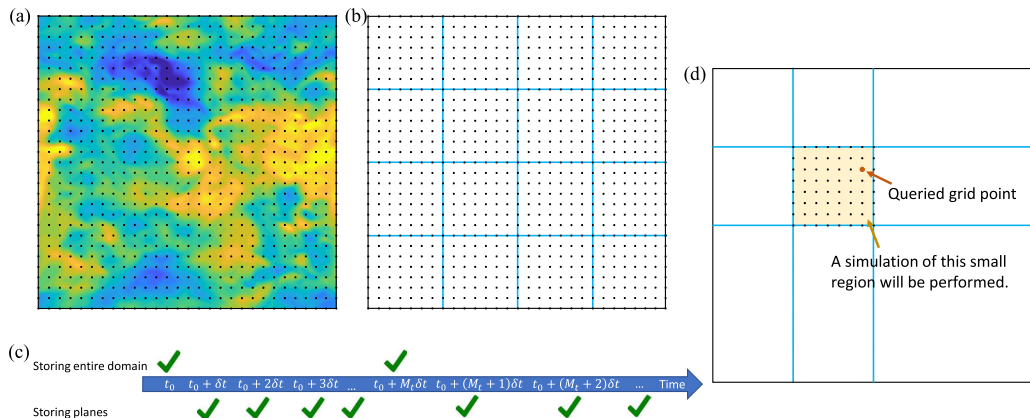


FIG. 1. Schematic of STSR. (a) Entire DNS domain containing a large ( $N^3$ ) number of grid points. (b) The entire DNS domain is divided into small cube regions by the blue lines. (c) The storage scheme of the spatiotemporal subsampling for resimulation. The data in the entire domain are stored at every  $M_t$  time step. The data on the planes (blue lines) and on the outer planes (black lines) are stored at every time step. (d) When data are required on grid points and time steps that are not stored in the database, a resimulation of a small region which includes the queried grid point is performed to obtain the data.

in Sec. III. The methodology is tested in the decaying isotropic turbulence, a well-understood and relatively simple flow described in Sec. III B. In Sec. IV, the influence of the boundary conditions on reproducibility of the simulations, up to the desired level of machine precision, is examined. The resimulation errors are studied in Sec. V in more detail, and their dependence on artificially introduced noise in boundary conditions is established in order to better understand requirements for reaching desired levels of accuracy, which are slightly relaxed from machine accuracy down to relative errors at the order of  $\approx 10^{-5}$  based on practical considerations. Section VI showcases an application using the recommended parameters. Finally, conclusions are presented in Sec. VII. The paper is limited to an account of the findings regarding methodology and requirements in the context of a simple flow at moderate computational scale. Construction of a large turbulence database system using the proposed STSR querying method is left as a future task.

## II. SUBSAMPLING AND LOCAL RESIMULATION

In this section, the basic concept of the proposed STSR approach is explained, together with an estimate of the data compression that can be achieved. Figure 1 is a two-dimensional (2D) schematic of a DNS domain and the storage scheme of the data to enable later resimulation. The flow domain inside the box in Fig. 1(a) represents the entire, or global, domain of the original simulation, e.g., from a simulation of isotropic turbulence, channel flow, boundary layer, etc. The global domain consists of a large number of grid points; in three dimensions, say,  $N^3 = N_x N_y N_z$ . By enforcing initial and boundary conditions on the global domain boundaries, the simulation is advanced forward in time, at a time step  $\delta t$ . The objective is to store a limited amount of data at each time step in order to enable resimulation of a subregion of the global domain. For this purpose, the global domain is divided into small subvolumes marked by the blue boundaries [Fig. 1(b)] corresponding to planes in a three-dimensional (3D) domain. For simplicity, the subvolumes here have the same shape and dimensions but the discussion and general results to be presented can be considered quite general. While the main simulation is performed, the state vector (i.e., velocity and pressure fields for incompressible flow) is stored on these planes. If the size of an individual resimulation subdomain is  $M_s$ , in three dimensions there will be  $3(N/M_s)$  such planes, each of size  $N^2$ .

Moreover, in order to limit the CPU cost of resimulation, after a number of time steps, the state vector data are stored at every grid point in the global domain. This occurs every  $M_t$  time steps, i.e., after a time equal to  $M_t \delta t$  [see Fig. 1(c)]. In the rest of this paper,  $t_n = n \delta t$  represents the physical time, while  $n$  represents the time step of the DNS. For a simulation lasting a total time  $T$ , the total number of full 3D fields to be stored is thus equal to  $\approx T/(M_t \delta t)$ .

After the direct simulation in the global domain has been completed and the subsampled data stored, data at a specific spatial and temporal location  $(\mathbf{x}, t)$  may be required, for example, to examine local flow states in particularly interesting subregions of the flow or to track particles through the flow. In general these locations do not correspond to stored data, and the data must be evaluated by reevaluating the flow evolution in the host subvolume and time interval [Fig. 1(d)].

Similar to the global domain, the flow in the resimulation subdomain is governed by the continuity and Navier-Stokes equations. The numerical solution requires the initial and boundary conditions. Suppose there exists an integer  $n$  such that  $[t_0 + nM_t \delta_{\text{DNS}} < t < t_0 + (n + 1)M_t \delta_{\text{DNS}}]$ , i.e., the time at which data are sought  $t$  lies between two instances where the entire global domain was stored. The data stored at  $(t_0 + nM_t \delta_{\text{DNS}})$  can then be used as the initial condition, and the plane data on the subdomain boundary that were stored at every time step between times  $(t_0 + nM_t \delta_{\text{DNS}})$  and  $t$  provide the boundary conditions needed for resimulation. Unless otherwise stated, the original simulation and its resimulation will adopt the same time step for forward integration of the governing equations.

To fix notation, in the rest of this paper the “global domain” refers to the domain of the original simulation [the black enclosing box in Fig. 1(d)]; a “subdomain” refers to the much smaller region containing a queried point or sets of points [the yellow region in Fig. 1(d)]; and “resimulation” refers to numerical solution of the governing equation in this subdomain using initial and boundary conditions extracted during the original computation and stored in the STSR database.

With the proposed approach, only a small fraction of data is stored and the fields can be reconstructed on demand from simulations within the small subregions. The data compression (inverse) ratio  $c$  can be estimated as

$$c \approx \frac{N^3 + 3N^2(N/M_s)(M_t - 1)}{N^3 M_t} = \frac{1}{M_t} \left( 1 - \frac{3}{M_s} \right) + \frac{3}{M_s}, \quad (1)$$

where  $N$  is the number of grid points in each direction in the entire domain.

Hence, if, for example,  $M_s = 128$  is used, and we store only every  $M_t = 200$  full 3D fields, the total storage requirement is about 2.8% of the original data. Performing the resimulation in the  $M_s^3$  subdomain is certainly much faster than doing a resimulation in the original full 3D volume: the CPU cost of resimulation is approximately  $M_t(12M_s^3 + M_s^3 \log_2 M_s)$ . Depending on the ratio of cost of storage and computation, as well as depending on patterns of data queries and usage, the optimal values of  $M_s$  and  $M_t$  could vary significantly. For now we simply observe that the  $8192^3$  grid database with  $\approx 10^4$  time steps mentioned in the introduction requiring over 80 PB of storage would require only about 2.2 PB if stored using subsampling with  $M_s = 128$  and  $M_t = 200$ , and the computational cost of the resimulation is only  $\mathcal{O}(10^{-6})$  of the cost of the full simulation.

The approach becomes particularly attractive in studies where only small subregions of the flow need to be interrogated later on. For example, in particle tracking studies, one only needs velocities in the immediate vicinity of particles to be used for interpolation. In other studies, researchers may want to zoom into areas where extreme events such as core of vortices or high dissipation take place. Or, one may wish to obtain a one-dimensional (1D) spectrum along some representative lines through the flow requiring data only along those lines rather than the entire domain. In such scenarios, storing the entire data or having to perform resimulation in the entire domain would be unnecessary and waste computational/storage resources.

One might consider that the present methodology is similar to data assimilation [20,22–24] or “nudging” [25] to deal with incorporation of incomplete and/or imperfect (noisy) data. In nudging, a penalization term is included in the Navier-Stokes equations, so that the resimulation result would

be pulled towards the original (observation) data. In that case, deviation in the initial condition is allowed, and the resimulation result will match the original data after several time steps, depending on initial condition, penalization term, flow condition, etc. However, the number of time steps needed for the resimulation to catch up to the original data is difficult to assess without using the correct initial condition. Therefore, for the purpose of reusing the DNS data, providing the correct initial condition becomes a necessary condition in this paper.

### III. NUMERICAL SCHEME AND FLOW CONFIGURATION

Incompressible flow of a Newtonian fluid satisfies the continuity and Navier-Stokes equations written here in skew-symmetric form:

$$\nabla \cdot \mathbf{u} = 0, \quad (2)$$

$$\frac{\partial \mathbf{u}}{\partial t} + \frac{1}{2}[\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) + (\mathbf{u} \cdot \nabla)\mathbf{u}] = -\nabla p + \nu \nabla^2 \mathbf{u}, \quad (3)$$

where  $\mathbf{u} = (u, v, w)^T$  is the velocity vector,  $t$  is time, and  $\nu$  is the fluid kinematic viscosity. The three velocity components  $u$ ,  $v$ , and  $w$  correspond to the  $x$ ,  $y$ , and  $z$  directions, respectively, and  $p$  is pressure divided by density. The advection term in Eq. (3) is expressed in the skew-symmetric form which conserves kinetic energy and reduces aliasing errors [26]. However, other forms of the advection term can also be adopted.

#### A. Temporal and spatial discretization

A  $\delta p$ -form prediction-correction algorithm [27–29] is used to decouple the velocity and pressure:

$$\frac{\mathbf{u}^* - \mathbf{u}^{(n-1)}}{\delta t} = -\text{Conv.} + \text{Diff.} - G(p^{(n-1)}), \quad (4)$$

$$DG\phi^{(n)} = \frac{D\mathbf{u}^*}{\delta t}, \quad (5)$$

$$\mathbf{u}^{(n)} = \mathbf{u}^* - \delta t G\phi^{(n)}, \quad (6)$$

$$p^{(n)} = p^{(n-1)} + \phi^{(n)}, \quad (7)$$

where  $\delta t$  is the time step, superscript  $(\cdot)^n$  denotes the  $n$ th step, Conv. is the discretized convective term, Diff. is the discretized diffusive term,  $G$  is the discretized gradient operator,  $D$  is the discretized divergence operator, and  $\phi$  is the pressure difference between two time steps. The advection term can be advanced in time explicitly using the explicit Euler or second-order Adams-Bashforth (AB2) scheme; the viscous term can be advanced using the Euler, AB2, or implicit Crank-Nicolson (CN) scheme.

A variant of the projection method referred to as the  $p$  form [30,31] ignores the pressure gradient term in the prediction step (4), and therefore  $\phi^{(n)}$  in the Poisson equation (5) is an approximation of the full pressure at the new time step, i.e.,  $p^{(n)} = \phi^{(n)}$ . A notable difference between the herein adopted  $\delta p$  and the  $p$  forms is in the boundary conditions: (i) the boundary condition of the elliptic pressure equation is the pressure difference in the  $\delta p$  form, and the pressure in the  $p$  form; (ii) in terms of the velocity, in order to ensure second-order accuracy, one should enforce  $\mathbf{u}^* = \mathbf{u}_\Gamma$  on the boundary of the computational domain  $\Gamma$  in the  $\delta p$  form, but  $\mathbf{u}^* = \mathbf{u}_\Gamma + \delta t G p_\Gamma^{(n-1)}$  in the  $p$  form. In the present paper, the  $\delta p$  form is adopted throughout. Although not presented here, use of the  $p$  form does not affect our results or conclusions.

A staggered grid [32] is used in order to avoid checkerboard pressure oscillations. The spatial derivatives are approximated with second-order central finite differences. In light of the computational cost of the pressure equation (5), it is important to ensure that the resimulation does

TABLE I. Statistics of decaying isotropic turbulence in the global domain ( $256^3$ ). The statistics are the same to within four digits for the five different time steps used, except for the quoted CFL numbers which are based on the case  $\delta t = 4 \times 10^{-3}$ .

Time $t$	rms velocity $u'$	Dissipation $\varepsilon$	Renumber $R_\lambda$	Kolmogorov scale $\eta$	CFL	
					$u' \delta t / \Delta x$	$u_{\max} \delta t / \Delta x$
0	0.6024	0.0770	113.24	0.01795	0.0982	0.4013
2	0.5185	0.0645	91.67	0.01876	0.0845	0.3699

not compromise any of the efficiency of the global solver. For instance, if the global domain is triply periodic, Fourier transform can be adopted in all three dimensions and the solution of (5) is inexpensive. The resimulation subdomain is, however, not periodic; we nonetheless adopt a fast Poisson solver using discrete sine and cosine transforms [33]. Details on the pressure Poisson solver used in resimulations are provided in Appendix A.

### B. Flow configuration: Decaying isotropic turbulence

The flow adopted in this paper as an example application of STSR is decaying isotropic turbulence in three dimensions. The global domain has dimensions  $2\pi \times 2\pi \times 2\pi$ , and is discretized uniformly using  $256^3$  grid points ( $N = 256$ ); the grid spacing is  $h = \Delta x = 0.02454$ . The domain is periodic in all three spatial directions. Time integration of the viscous and convective terms starts with one Euler step at the initial condition, and is subsequently evolved using AB2. A snapshot from a  $1024^3$  isotropic turbulence data set [12] in JHTDB is used as the initial condition, subsampled every four grid points. After a transient of a few hundred time steps, all velocity and pressure fields are stored and designated as the initial condition ( $t = 0, n = 0$ ) of our set of numerical experiments.

The kinematic viscosity is set to  $\nu = 2 \times 10^{-3}$  in order to provide appropriate resolution of the viscous scale at the initial time. Five different time steps will be used,  $\delta t = \{4, 2, 1, 0.5, 0.25\} \times 10^{-3}$ . Simulations are advanced from  $t = 0$  to 2. Some basic statistics of the simulation of this decaying isotropic turbulence are listed in Table I. These were verified to be accurate to within four digits for the various choices of the time step; the reported CFL values are based on the largest  $\delta t = 4 \times 10^{-3}$ . The kinetic energy and dissipation spectra are shown in Fig. 2. The dissipation spectra are displayed in Kolmogorov units, showing that the simulation is very well resolved in space (note that the spatial resolution is much better than in the JHTDB original data even if using fewer points since here we simulate a much lower Reynolds number with a much higher  $\nu$ ).

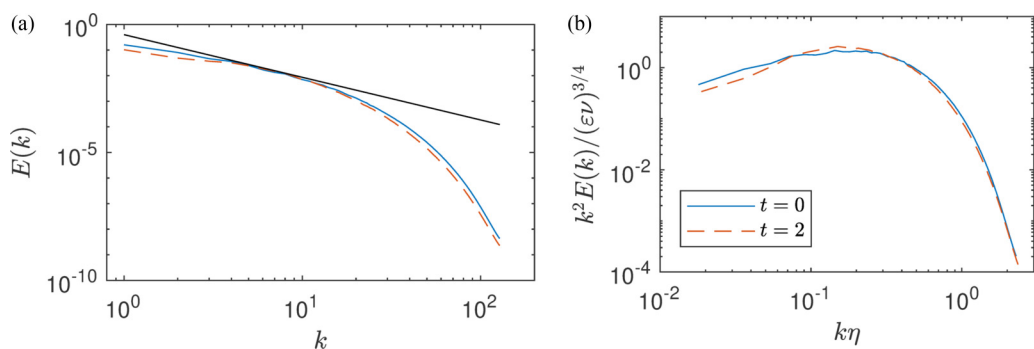


FIG. 2. Radial (a) kinetic energy and (b) dissipation spectra at the start of the simulation  $t = 0$  and the end of the simulation  $t = 2$ . The black straight line in (a) has a slope of  $-5/3$ .



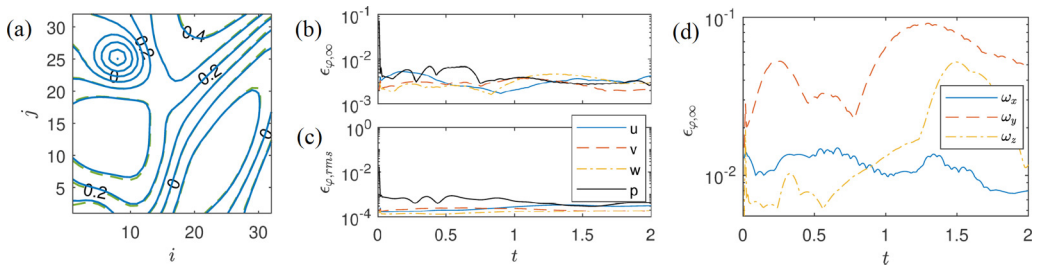


FIG. 3. (a) Contour plot of pressure distribution on a randomly selected slice in the  $32^3$  subdomain resimulation at a randomly selected, representative, time step. The dash contour lines are the original simulation, while the solid contour lines are the resimulation. (b)  $\epsilon_{\varphi,\infty}$  as a function of time  $t$ . (c) rms error  $\epsilon_{\varphi,\text{rms}}$  as a function of time  $t$ . In the resimulation, the velocity boundary conditions are  $\mathbf{u}$  and the pressure boundary condition is Neumann type. (d)  $L^\infty$  vorticity errors as a function of time. All plots are for the case  $\delta t = 4 \times 10^{-3}$ .

While performing the simulation in the global domain, data are stored at every time step to be used for later analysis and comparison with resimulation results. For the sample resimulations and numerical experiments to be described in the next section, a subdomain consisting of  $32^3$  grid points is selected (i.e.,  $M_s = 32$ ) located at a random location within the global domain. To compare results from resimulation to the original global domain simulation, a normalized local error is defined according to

$$\epsilon_\varphi(x, y, z, t) = \frac{|\varphi_{\text{OS}}(x, y, z, t) - \varphi_{\text{RS}}(x, y, z, t)|}{\text{rms}(\varphi_{\text{OS}})}, \quad (8)$$

where  $\varphi$  could be any quantities, such as  $u$ ,  $v$ ,  $w$ ,  $p$ , or vorticity components  $\omega_i$ ;  $\text{rms}(\cdot)$  is the root-mean-square (rms) value within the subdomain; OS refers to the original simulation; and RS refers to the resimulation. We focus on the  $L^\infty$  errors evaluated as a function of time within the subdomain,  $\epsilon_{\varphi,\infty}(t)$ , which is a stringent upper bound on the resimulation errors.

#### IV. PRELIMINARY RESULTS

As a first test we consider a resimulation starting from the initial condition at  $t = 0$ . One can use the velocity and pressure fields at  $n = 0$  as the resimulation initial condition. The boundary conditions at time step  $n$  are  $\mathbf{u}_{rs,\Gamma}^{(n)} = \mathbf{u}_{os,\Gamma}^{(n)}$  for velocity and  $(\partial\phi_{\text{RS}}^{(n)}/\partial\mathbf{n})_\Gamma = (\partial\phi_{\text{OS}}^{(n)}/\partial\mathbf{n})_\Gamma$  for pressure increment. Above,  $\mathbf{n}$  denotes the outward pointing normal unit vector to the boundary  $\Gamma$  (distinct from time step  $n$ ).

Using these initial and boundary conditions, the resimulation is integrated in time between  $t = 0$  and  $t = 2$  (i.e., for 500 time steps for the case  $\delta t = 4 \times 10^{-3}$ ). Figure 3(a) shows a comparison of the pressure distribution on a representative plane and time. While overall the agreement may appear good, there are some noticeable differences especially near the lower left and upper right boundaries.

More quantitatively, the maximum error ( $L^\infty$ ) and rms error over the subdomain are shown as functions of time in Figs. 3(b)–3(d). The error is large already at the first resimulation time step and then remains at similar order of magnitude. The  $L^\infty$  and rms errors of velocity and pressure are of order  $10^{-3}$ , and vorticity errors are about one order of magnitude higher and could reach near 10%; these errors are too large compared to our stated desired level of accuracy. (We have found the vorticity errors are typically one order of magnitude higher than velocity, so we only show vorticity results towards the end when showing results of acceptable levels of accuracy.)

An interesting observation is that the errors do not grow exponentially, suggesting that the observed errors are not caused by chaotic dynamics as one may have initially suspected based

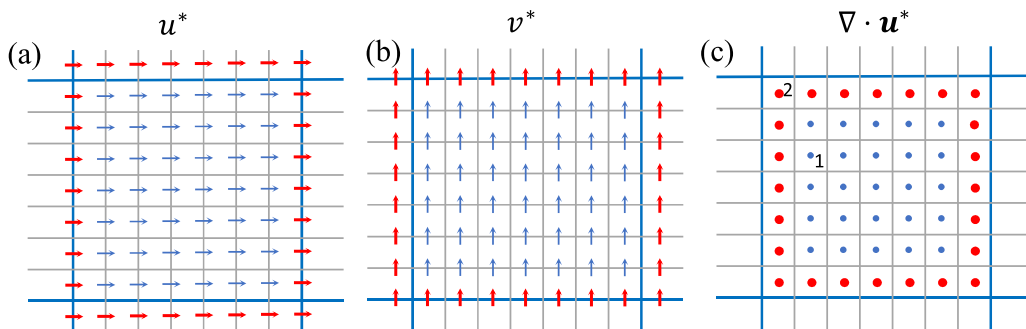


FIG. 4. Comparisons of (a)  $u^*$ , (b)  $v^*$ , and (c)  $\nabla \cdot \mathbf{u}^*$  between the resimulation and the original simulation. Blue (thin) and red (thick) symbols denote the quantities in resimulation match/mismatch to the original simulation data.

on the nonlinear character of the governing equations. The reason might be that the subdomain size is relatively small so that even if there are differences between the two fields the resimulation dynamics are slaved to the original dynamics by the imposed boundary conditions.

Naturally we anticipate that if the subdomain was sufficiently large simply providing boundary conditions would not guarantee that the two trajectories would not diverge eventually in time due to chaotic dynamics in the domain interior. Regardless of the origin of the observed errors, we have experimented with a number of parameters such as the time step and spatial resolution, and the basic conclusion remains that the errors are significant and far from the desired accuracy for our database application. Aiming to reduce these errors, we analyze the source of the discrepancy and identify the appropriate choice of implementing initial and boundary conditions in order to greatly reduce these errors.

#### A. Resimulation boundary conditions: $u$ versus $u^*$

Consider the resimulation procedure from the initial condition  $n = 0$  to the first time step  $n = 1$ .

At time step  $n = 0$ , the initial conditions are based on  $\mathbf{u}_{\text{OS}}^{(0)}$  and  $p_{\text{OS}}^{(0)}$  of the global computation, and therefore the resimulation matches that state exactly.

Since  $\mathbf{u}_{\text{RS}}^{(0)}$  and  $p_{\text{RS}}^{(0)}$  match the global simulation, the convective, diffusive, and pressure gradient terms *inside* the resimulation subdomain are correct.

Because the momentum equations are both integrated with an Euler method in the original simulation and the resimulation to the first time step  $n = 1$ ,  $\mathbf{u}^*$  *inside* the subdomain is the same as in the original simulation [blue thin arrows in Figs. 4(a) and 4(b)]. Meanwhile,  $\mathbf{u}_{\text{OS}}^{(1)}$  on  $\Gamma$  are applied as the velocity boundary conditions. However, the data on and outside the subdomain boundary are also  $\mathbf{u}^*$  in the original simulation, since they lie within the global domain. Thus, the resimulation does not match the original computation on and outside the subdomain boundary [red thick arrows in Figs. 4(a) and 4(b)].

The source term of the Poisson equation is then computed, and the comparison with the original simulation is shown in Fig. 4(c). Considering two grid points as examples, the source term at point 1 is calculated from surrounding values of  $\mathbf{u}_{\text{RS}}^*$ , all of which are identical to the original simulation. Thus the source term is correct (blue small dots). However, at point 2, the values of  $u^*$  at left and  $v^*$  above are different from the original simulation, thus the source term at this grid point differs from the global solver (red big dots).

The Poisson equation with perturbed source term is solved and  $\mathbf{u}_{\text{RS}}$  and  $\phi_{\text{RS}}$  therefore contain errors.



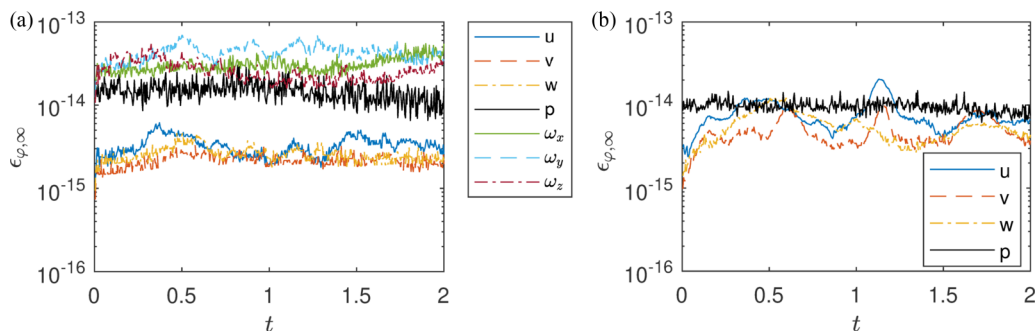


FIG. 5. Errors  $\varepsilon_{\phi,\infty}$  as a function of time (for the case  $\delta t = 4 \times 10^{-3}$ ). In the resimulation, the velocity boundary conditions are  $\mathbf{u}^*$ , and the pressure boundary condition is (a) Neumann type and (b) Dirichlet type.

The above discussion shows that the choice of velocity boundary conditions leads to errors in the resimulation outcome, as reported in Fig. 3. The remedy is to adopt  $\mathbf{u}_{OS}^*$  as the velocity boundary condition in the resimulation procedure.

Thus, switching procedures, now the values of  $\mathbf{u}_{OS}^*$  at the boundaries of the subdomains were stored during the global simulation. These were subsequently used for boundary conditions in the local resimulation procedure. The resulting  $L^\infty$  errors are reported in Fig. 5(a). Indeed the resimulation velocities and pressure agree with the global computation results exactly, to within machine precision.

References [34] and [35] showed that Dirichlet and Neumann pressure boundary conditions are equivalent, to within a constant. We confirmed the same behavior for the resimulations by performing a test with pressure Dirichlet boundary conditions  $\phi_{RS} = \phi_{OS}$  on the subdomain boundary  $\Gamma$ . The resimulation errors, shown in Fig. 5(b), are still at machine accuracy, the same as those in the resimulations with the pressure Neumann boundary conditions (note that in both cases  $\mathbf{u}_{RS}^* = \mathbf{u}_{OS}^*$  is enforced on  $\Gamma$ ).

### B. Crank-Nicolson scheme

In simulations of nonhomogeneous flows such as wall-bounded turbulent flows, the viscous term may limit the time step due to the stability restriction. Therefore, this term is often discretized in time using the CN scheme in order to mitigate the stability restriction. Using CN, Eq. (4) is approximated with the alternating direction implicit (ADI) method according to

$$(1 - A_x)(1 - A_y)(1 - A_z)\mathbf{u}^* = \delta t \left[ -\text{Conv.} + \frac{1}{2}vL(\mathbf{u}^{(n-1)}) - G(p^{(n-1)}) \right] + \mathbf{u}^{(n-1)}, \quad (9)$$

where  $A_x = \frac{1}{2}v\delta t L_x$ ,  $A_y = \frac{1}{2}v\delta t L_y$ ,  $A_z = \frac{1}{2}v\delta t L_z$ ,  $\text{Conv.} = \alpha_c C(\mathbf{u}^{(n-1)}) + \beta_c C(\mathbf{u}^{(n-2)})$  is the integrated advection term,  $L$  is the discretized Laplacian operator, and  $L_x$ ,  $L_y$ , and  $L_z$  are the discretized Laplacian operators in the  $x$ ,  $y$ , and  $z$  directions. The procedure for solving the above equation consists of evaluating  $\mathbf{u}^*$  in each of the three directions successively.

(i) Solve for  $\mathbf{u}^{*1}$  in the  $x$  direction, where  $(1 - A_x)\mathbf{u}^{*1}$  equals the right-hand side of Eq. (9) with  $x$  boundary conditions.

(ii) Solve for  $\mathbf{u}^{*2}$  in the  $y$  direction, where  $(1 - A_y)\mathbf{u}^{*2} = \mathbf{u}^{*1}$  with  $y$  boundary conditions.

(iii) Solve for  $\mathbf{u}^* = \mathbf{u}^{*3}$  in the  $z$  direction, where  $(1 - A_z)\mathbf{u}^{*3} = \mathbf{u}^{*2}$  with  $z$  boundary conditions.

In Sec. IV A, it was demonstrated that  $\mathbf{u}^*$  should be the velocity boundary condition if both the original and resimulation algorithms are explicit Euler/AB2. When CN/ADI is adopted, however, different intermediate velocity boundary conditions are required. Specifically,  $\mathbf{u}^{*1}$  should be applied on the boundaries during the inversion of the  $x$ -diffusion term,  $\mathbf{u}^{*2}$  should be applied on the boundaries during the solution in the  $y$  direction, and  $\mathbf{u}^* = \mathbf{u}^{*3}$  should be applied on the boundaries in the final  $z$  direction.

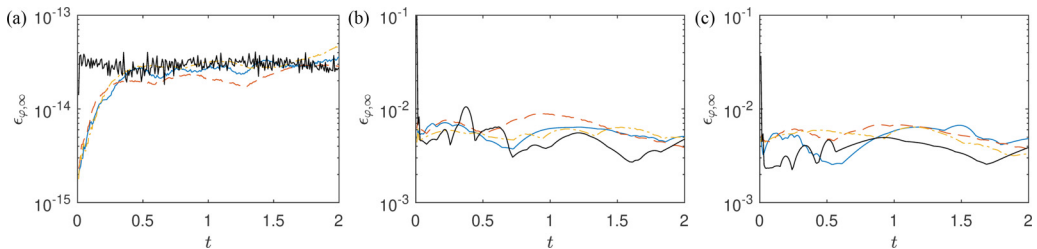


FIG. 6. Resimulation errors with different velocity boundary conditions, which are (a)  $\mathbf{u}^{*1}$ ,  $\mathbf{u}^{*2}$ , and  $\mathbf{u}^{*3}$  in the corresponding directions; (b)  $\mathbf{u}$  in all directions; and (c)  $\mathbf{u}^* = \mathbf{u}^{*3}$  in all directions. In all plots,  $\Delta t = 4 \times 10^{-3}$ . See Fig. 5 for legend.

We demonstrate this requirement by performing the original/global simulation and the resimulation using the CN scheme as described above, and compare the results with cases in which some of the specific directional requirements for  $\mathbf{u}^*$  are relaxed. The resimulation errors with the correct boundary-condition implementation are shown in Fig. 6(a). The resimulation errors remain near  $10^{-14}$  for all velocities and pressure. As comparison, the resimulations with either  $\mathbf{u}$  or  $\mathbf{u}^* = \mathbf{u}^{*3}$  (the last step of the ADI velocity boundary conditions are also performed. Both produce significant error levels, between  $10^{-3}$  and  $10^{-2}$  [Figs. 6(b) and 6(c)].

## V. ANALYSIS OF DOMINANT SOURCES OF ERRORS

In Sec. IV A, the correct velocity boundary conditions for resimulation were shown to be  $\mathbf{u}^*$ . It was shown that using  $\mathbf{u}^*$  on the boundaries based on surface data stored at every DNS time step, and replicating the precise time advancement scheme at every time step between the original DNS and the resimulation, yielded machine accuracy from resimulation. However, in practical applications of STSR, one may wish to relax some of these requirements. For example, one may wish to store the boundary values not at every time step and use moderate subsampling (e.g., snapshots of the  $1024^3$  isotropic turbulence data set in JHTDB are stored only every ten simulation steps, and temporal polynomial interpolation is used to find data between stored time steps). Or, one may wish to use a different time-advancement scheme during the initial time stepping of the resimulation. Each of these approaches will induce some additional error and prevent the resimulation to reach machine precision. In order to establish a clear understanding of these errors, it is useful to quantify the amplification of errors by the resimulation procedures.

In order to lay the foundation for the subsequent discussions, we intentionally add noise to the boundary-condition values  $\mathbf{u}^*$ . We use zero-mean Gaussian white noise and define the contaminated boundary condition on the boundary  $\Gamma$ , for example, for the  $u$  component, as

$$u_\sigma^* = u^*[1 + \sigma\mathcal{N}(0, 1)], \quad (10)$$

where  $\sigma$  represents the rms of the added noise as multiple of the original signal. Moreover,  $\mathcal{N}(0, 1)$  is the standard normal distribution with zero mean and unit variance. Similar noise perturbations are added to the two other components  $v^*$  and  $w^*$ , and pressure increment  $\phi$ , at all time steps  $n > 0$ .

Resimulation experiments are performed for four different levels of  $\sigma$  ( $10^{-4}$ – $10^{-10}$ ) using  $\mathbf{u}_\sigma^*$  and  $\partial\phi_\sigma/\partial\mathbf{n}$  as boundary conditions. The resimulation errors  $\varepsilon_{u,\infty}$  are shown in Fig. 7(a) as a function of  $t$  with different noise levels  $\sigma$ ; only  $u$  errors are plotted for clarity. Although the noise levels are different, the errors are qualitatively similar at different values of  $\sigma$  and only differ in magnitude. Figure 7(b) shows the scaling of  $\max_t[\varepsilon_{\phi,\infty}]$  with  $\sigma$ . The results clearly show that resimulation errors grow linearly with the magnitude of the added noise level in the boundary conditions.

It should be noted that, in the above analysis, the noise is added to the boundary conditions at all time steps after the initial condition, i.e.,  $n \geq 1$ , and the resimulation errors are proportional to

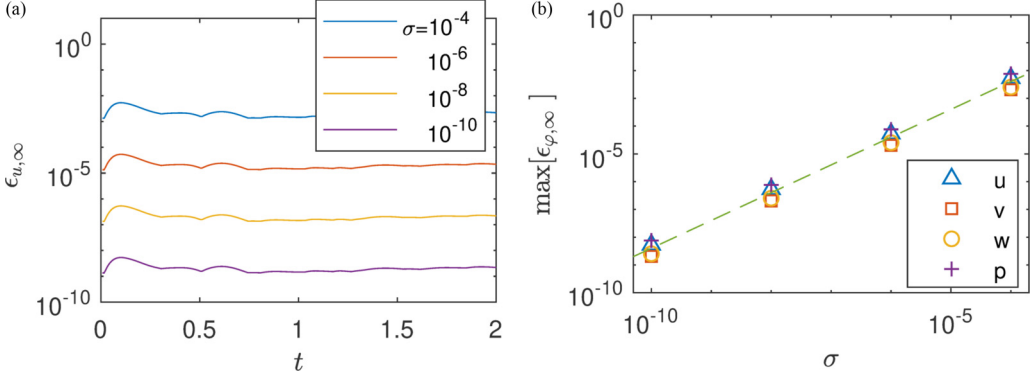


FIG. 7. Resimulations with different levels of noises added to the velocity boundary conditions  $\mathbf{u}^*$ . (a) Resimulation errors  $\epsilon_{u,\infty}$  against  $t$ . Only  $u$  errors are plotted for clarity. It has been checked that  $v$ ,  $w$ , and  $p$  errors behave similarly. (b)  $\max[\epsilon_{\varphi,\infty}(t > 0)]$  as a function of  $\sigma$ . The dashed line has a slope of 1. In both plots,  $\delta t = 4 \times 10^{-3}$ .

the input errors. If the noise is added at the initial condition across the entire resimulation domain at  $n = 0$ , similar results are obtained (not shown here).

#### A. Reexamination of $u$ boundary-condition errors

We have seen that the resimulation errors are proportional to the input errors. We now revisit the errors discussed in Sec. IV A, where we first naively applied  $\mathbf{u}$  as the velocity boundary conditions, to explain the observed errors based on the findings that errors are linearly proportional to boundary-condition errors.

From Eq. (6), one can easily show that the difference between  $\mathbf{u}^{(n)}$  and  $\mathbf{u}^*$  is second order in time:

$$\mathbf{u}^{(n)} - \mathbf{u}^* = -\delta t \nabla \phi^{(n)} = -\delta t \nabla (p^{(n)} - p^{(n-1)}) \sim -(\delta t)^2 \nabla \left( \frac{\partial p}{\partial t} \right). \quad (11)$$

Based on the results in Fig. 7, one would then expect that applying  $\mathbf{u}$  as boundary conditions in the resimulation would lead to second-order errors in  $\delta t$ . This expectation was tested by performing the global and resimulations with different values of  $\delta t$  and prescribing  $\mathbf{u}$  as the velocity boundary condition in the resimulations. The resulting resimulation errors are plotted in Figs. 8(a) and 8(b). As in Fig. 7(a),  $\epsilon_{\varphi,\infty}$  behave qualitatively similar for different values of  $\delta t$ . The maximum errors,  $\max_t[\epsilon_{\varphi,\infty}]$ , are reported in Fig. 8(c). Surprisingly, the pressure errors are only first order in  $\delta t$ , while the velocity errors are second order, as expected. In addition, we find that the pressure errors recover second-order accuracy at  $n > 1$  [Fig. 8(d)]. In fact, Figs. 8(c) and 8(d) show that the maximum pressure errors are first order in  $\delta t$  for  $n \geq 1$ , but second order for  $n > 1$ . This observation suggests that the pressure errors are of first order at  $n = 1$  but second order afterwards. The inset of Fig. 8(b) shows the pressure errors near  $n = 0$ , while Fig. 9 shows the  $u$  and  $p$  errors along a line in the center of the subdomain.

A brief explanation follows: assume the initial field of the resimulation matches the original global computation. In the first time step, if  $\mathbf{u}_{\text{OS}}^{(1)}$  is used as the velocity boundary condition, i.e.,  $\mathbf{u}_{\Gamma}^* = \mathbf{u}_{\text{OS}}^{(1)}$ , the subdomain now contains  $\mathcal{O}(\delta t^2)$  errors at the boundaries:

$$\epsilon(\mathbf{u}^*) = \begin{cases} \mathbf{u}^{(1)} - \mathbf{u}^* = (\delta t)^2 \frac{\partial}{\partial x} \left( \frac{\partial p}{\partial t} \right) |_{n=1} = \delta t^2 \zeta_{n=1} & \text{on the boundaries} \\ 0 & \text{inside the subdomain} \end{cases}, \quad (12)$$

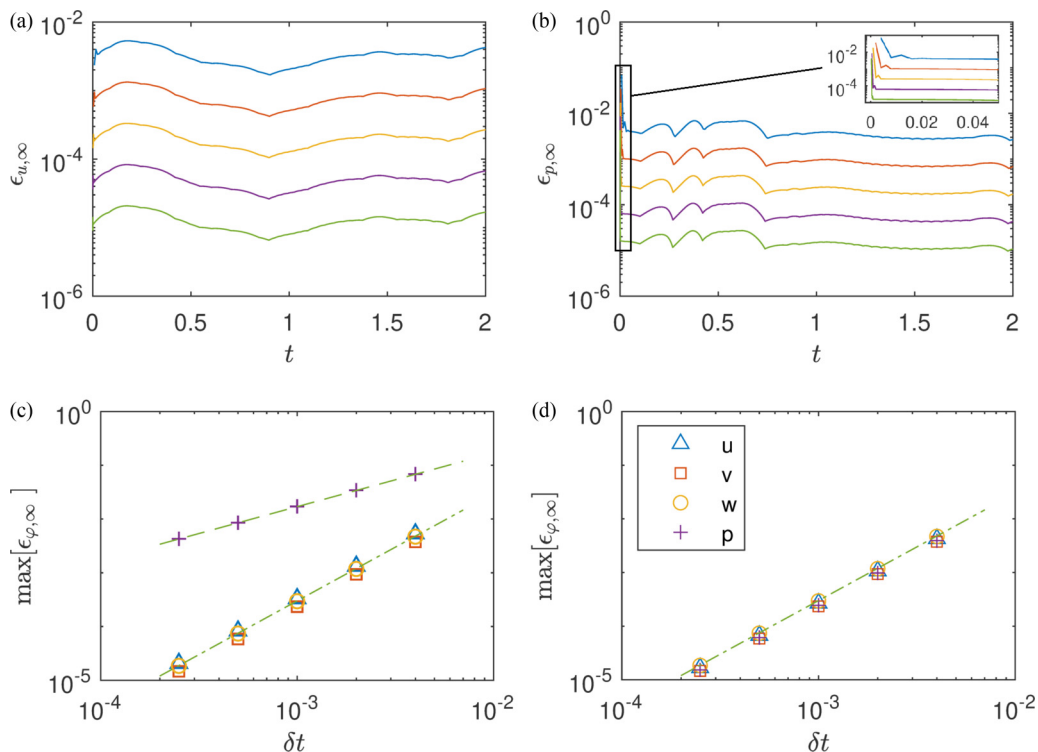


FIG. 8. Resimulations with  $\mathbf{u}$  as the velocity boundary conditions using different time steps. (a)  $u$  errors  $\varepsilon_{u,\infty}$  as a function of time,  $t$ . (b) Pressure errors  $\varepsilon_{p,\infty}$  as a function of time  $t$ . The inset is a zoom near  $t = 0$ . In (a) and (b), lines from top to bottom represent simulations with  $\delta t = 4 \times 10^{-3}$ ,  $2 \times 10^{-3}$ ,  $1 \times 10^{-3}$ ,  $5 \times 10^{-4}$ , and  $2.5 \times 10^{-4}$ , respectively. (c)  $\max[\varepsilon_{\varphi,\infty}(n \geq 1)]$  as a function of  $\delta t$ . (d)  $\max[\varepsilon_{\varphi,\infty}(n > 1)]$  as a function of  $\delta t$ . In (c) and (d), the dashed line has a slope of 1 and the dash-dotted line has a slope of 2.

where  $\zeta = \frac{\partial}{\partial x}(\frac{\partial p}{\partial t})$ . From the right-hand side of Eq. (5) and Fig. 4, the source term of the Poisson equation will therefore have  $\mathcal{O}(\delta t)$  errors due to the errors at the subdomain boundaries:

$$DG\epsilon(\phi^{(1)}) = \frac{D\epsilon(\mathbf{u}^*)}{\delta t} = \begin{cases} \delta t^2 \zeta_{n=1}/h\delta t = \delta t \zeta_{n=1}/h & \text{on the boundaries} \\ 0 & \text{inside the subdomain} \end{cases} \quad (13)$$

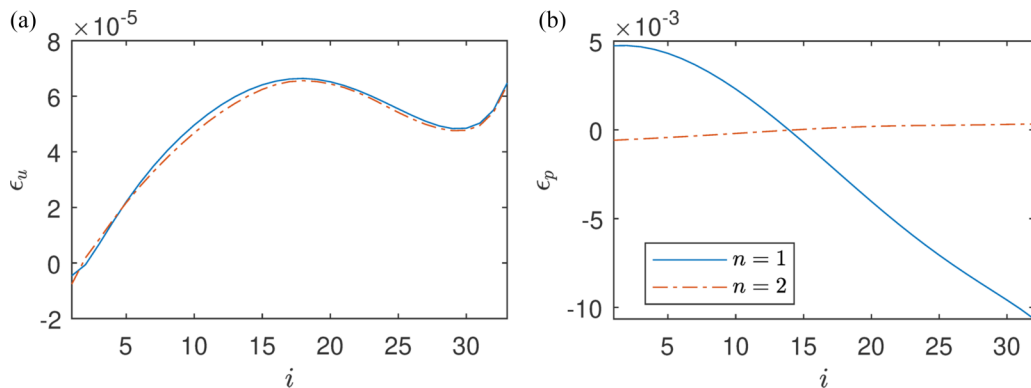


FIG. 9. Relative errors of (a)  $u$  and (b)  $p$  along a line in the center of the subdomain at the first (solid) and second (dash-dotted) time step in the  $\delta t = 4 \times 10^{-3}$  case.

Even though the nonzero source terms only exist at the boundary nodes in Eq. (13), the errors in  $\phi$  contaminate the entire subdomain due to the ellipticity of the Poisson operator. Thus  $\phi$  errors, as well as  $p$  errors, are  $h\delta t\zeta_{n=1} = \mathcal{O}(\delta t)$  at  $n = 1$ . It is important to note here that  $\epsilon(\phi^{(1)})$  is linearly distributed in the subdomain [can be verified analytically to be a solution of Eq. (13), or refer to Fig. 9(b)]. As a result, the gradient of  $\epsilon(\phi^{(1)})$  is uniform in the correction step, leading to a uniform  $\delta t^2\zeta_{n=1}$  error in the velocity within the subdomain:

$$\epsilon(\mathbf{u}^1) = \epsilon(\mathbf{u}^*) - \delta t G\epsilon(\phi^1) = \delta t^2\zeta_{n=1} = \mathcal{O}(\delta t^2). \quad (14)$$

At the second time step  $n = 2$ ,  $\mathbf{u}^*$  have uniform  $\mathcal{O}(\delta t^2)$  errors both inside the subdomain and on the boundaries: the errors inside the subdomain,  $\delta t^2\zeta_{n=1}$ , come from  $\mathbf{u}^{(1)}$  (see above), while the errors on the boundaries,  $\delta t^2\zeta_{n=2}$ , come from the new velocity boundary conditions. The leading  $\mathcal{O}(\delta t^2)$  errors of  $\mathbf{u}^*$  are canceled out during the calculation of the divergence of  $\mathbf{u}^*$ ,

$$D\epsilon(\mathbf{u}^*) = \begin{cases} \delta t^2\zeta_{n=2} - \delta t^2\zeta_{n=1} = \delta t^3 \frac{\partial \zeta}{\partial t} \Big|_{n=1} & \text{on the boundaries} \\ \delta t^2\zeta_{n=1} - \delta t^2\zeta_{n=1} = \mathcal{O}(\delta t^3) & \text{inside the subdomain} \end{cases}, \quad (15)$$

leading to second-order errors in the source term of the Poisson equation, also in the pressure field at  $n = 2$ . In addition, the velocity errors remain at second order:

$$\epsilon(\mathbf{u}^{(2)}) = \epsilon(\mathbf{u}^*) - \delta t G\epsilon(\phi^{(2)}) = \mathcal{O}(\delta t^2) - \delta t \mathcal{O}(\delta t^2) = \mathcal{O}(\delta t^2). \quad (16)$$

The preceding analysis thus demonstrates that the observed errors when using  $\mathbf{u}$  instead of  $\mathbf{u}^*$  as boundary conditions for resimulation scale in expected ways with the size of time step. If one wanted to use  $\mathbf{u}$  instead of  $\mathbf{u}^*$  for resimulation, however, the required time steps would be too small to be practical for purposes of the STSR.

### B. Errors from mismatch in temporal discretization

The above results all assumed that the resimulation starts from an Euler scheme, same as the original computation which at  $n = 0$  also began using an Euler step. This ensures that the resimulation could calculate the intermediate velocity inside the subdomain correctly as seen in Fig. 4, and reproduce the original simulation data precisely, when using the  $\mathbf{u}_{OS}^*$  boundary conditions.

However, in applications of STSR, the resimulation will typically start at any of the stored original simulation time steps, i.e., when  $n$  equals any integer multiple of  $M_t\delta t$ . Recall that the original simulation used AB2 time stepping at those times, not Euler. As a result, for the resimulation to reproduce the original computation, it must adopt an AB2 scheme from its start. However, this requirement can only be met if two consecutive time steps are stored to be used as initial condition. Otherwise, with a single field, the resimulation must adopt a first Euler step and will therefore deviate from the original AB2-based computation.

In order to demonstrate the errors incurred by an initial Euler step, we perform the following experiment: the data on the entire domain are stored at  $t = 1$ , meaning the initial condition for the resimulation is now  $\mathbf{u}_{OS}$  and  $p_{OS}$  at  $t = 1$ . The resimulation starts there with a single Euler scheme and then continues with AB2.

At the first time step after the initial condition, the Euler scheme will introduce local truncation errors of  $\mathcal{O}(\delta t^2)$  into the resimulation. The resimulation errors are shown in Fig. 10. Similar to the case which uses  $\mathbf{u}$  as the velocity boundary condition (Sec. V A), the  $p$  errors are first order in  $\delta t$  at the first time step, but second order afterwards. On the other hand, velocity errors are always second order.

In addition, we considered another case to explore errors incurred if the time stepping scheme used in the resimulation is always different from that in the original one. We performed resimulation with the Euler scheme from  $t = 1$  and for all time steps, rather than for the first step only. In this case, the Euler scheme has global errors of  $\mathcal{O}(\delta t)$  compared to AB2. The errors are shown in Fig. 11. The  $\mathbf{u}$  errors increase over  $t$ . This is due to the cumulative effect of the local truncation errors committed

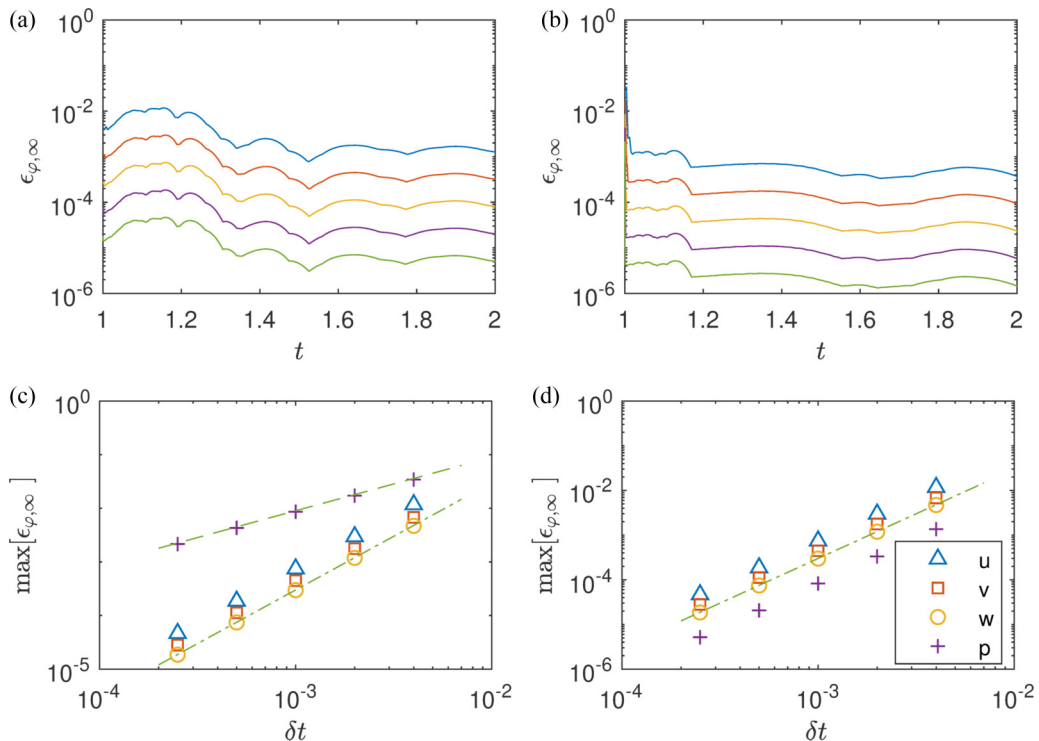


FIG. 10. Resimulation error evolution when using an Euler scheme at the first time step and then continuing with AB2 ( $1 \leq t \leq 2$ ). The original simulation used the AB2 scheme. (a)  $u$  error  $\varepsilon_{u,\infty}$  against  $t$ . (b)  $p$  error  $\varepsilon_{p,\infty}$  against  $t$ . In (a) and (b), lines from top to bottom represent simulations with  $\delta t = \{4, 2, 1, 0.5, 0.25\} \times 10^{-3}$ , respectively. (c)  $\varepsilon_{\phi,\infty}$  against  $\delta t$  at the first time step. (d)  $\max[\varepsilon_{\phi,\infty}]$  against  $\delta t$  after the first time step. In (c) and (d), the dashed line has a slope of 1 and the dash-dotted line has a slope of 2.

in each step from the Euler scheme. As a result, the velocity errors grow from second order to first order [see Figs. 11(c) and 11(d)]. On the other hand, the  $p$  errors are already first order at the first time step, and retain that scaling, consistent with Euler's global truncation errors  $\mathcal{O}(\delta t)$ .

### C. Errors from temporal substepping

In the previous section, it was shown that the resimulation has  $\mathcal{O}(\delta t^2)$  errors if started with an Euler scheme at an arbitrary time. These errors are too large for reproducing a DNS database using realistic values of  $\delta t$ . For example, when  $\delta t = 4 \times 10^{-3}$ , even if we discard the results at the first time step, the relative errors between the original and resimulation are approximately  $10^{-3}$ – $10^{-2}$  in subsequent time steps. Using an initial Euler step in the resimulation compared to AB2 in the original computation results in an initial error that persists in time—consistent with the behavior when artificial errors were included in the initial conditions. Although one could store an extra snapshot so that the resimulation starts with AB2 and obtain error-free data, this approach would appreciably increase the storage requirements.

Rather than storing two time steps, we examine a different approach that does not increase the required storage but only increases CPU cost during resimulation: temporal substepping.

This idea aims to minimize the error between the original single AB2 step and many smaller steps the first of which is Euler followed by AB2.

Consider integration from  $t$  to  $t + \delta t$ . The analytic integration could be approximated by an AB2 scheme or an Euler scheme both with a time step size  $\delta t$ . We have already seen in the previous



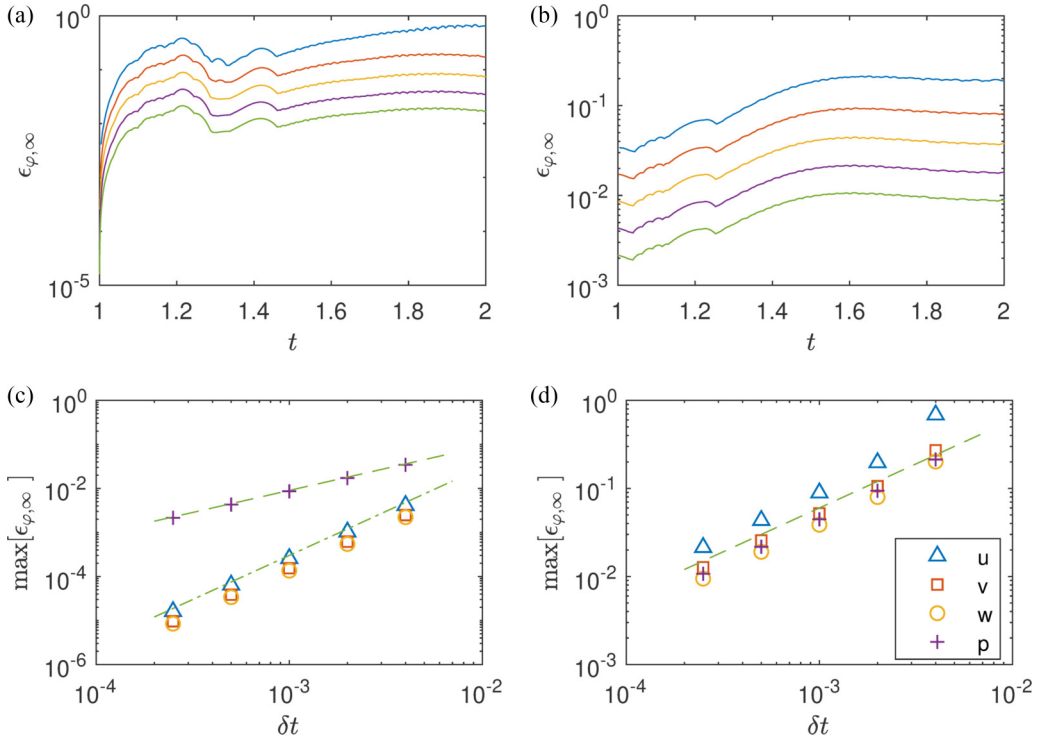


FIG. 11. Resimulations using and Euler time advancement throughout ( $1 \leq t \leq 2$ ). The original simulation always uses the AB2 scheme. (a)  $u$  error,  $\epsilon_{u, \infty}$ , against  $t$ . (b)  $p$  error,  $\epsilon_{p, \infty}$ , against  $t$ . In (a) and (b), lines from top to bottom represent simulations with  $\delta t = \{4, 2, 1, 0.5, 0.25\} \times 10^{-3}$ , respectively. (c)  $\epsilon_{\varphi, \infty}$  against  $\delta t$  at the first time step. (d)  $\max[\epsilon_{\varphi, \infty}]$  against  $\delta t$  after the first time step. In (c) and (d), the dashed line has a slope of 1 and the dash-dotted line has a slope of 2.

section that the differences between AB2 and Euler schemes lead to resimulation errors. Usually, an AB2 scheme produces smaller errors than Euler compared with analytic (true) values. On the other hand, the time step from  $t$  to  $t + \delta t$  could also be divided into, say,  $k$  subtime steps: the size of each subtime step is thus  $\delta t/k$  [see Fig. 12 for an example with  $k = 4$ ]. Integration from  $t$  to  $t + \delta t$  would then be computed using Euler in the first subtime step, then AB2 in the remaining  $(k - 1)$  subtime steps. The numerical integration results will approach the true value with increasing number of substeps  $k$ . The single full-time-step Euler integration is the special case with  $k = 1$ . Thus, one could expect that the errors between the single full-time-step AB2 integration and the integration with temporal substepping would decrease first, then increase, and finally reach an asymptotic value as the number of time substeps  $k$  increases: the asymptotic value is the errors of the AB2 scheme itself. Ideally, there will be a  $k$  with which the resimulation errors are minimized, even though this optimized  $k$ , if it exists, would be different from one simulation to another.

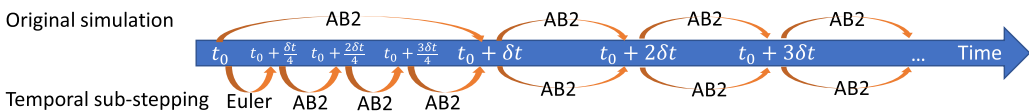


FIG. 12. Schematic of temporal substepping with four subtime steps.

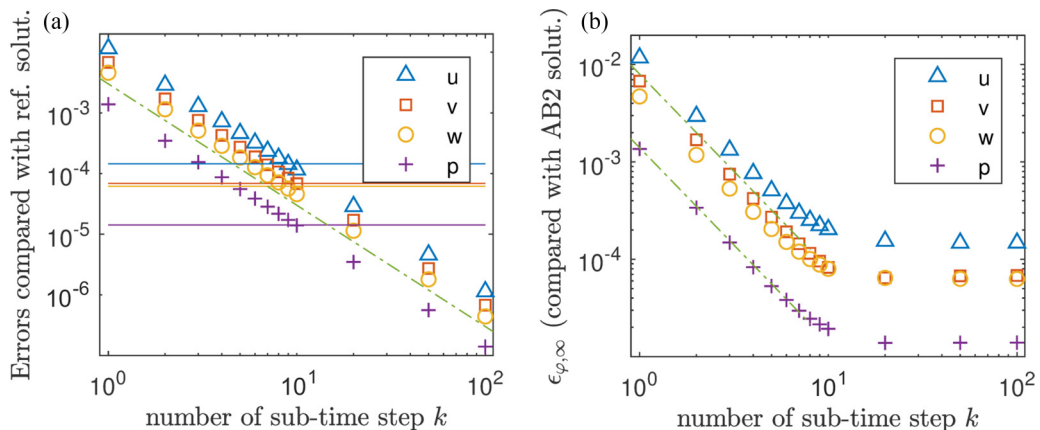


FIG. 13. (a) The  $L^\infty$  relative errors compared with the reference resimulation ( $k = 1000$ ). The symbols represent the resimulations with subtime steps, while the lines represent the original simulation with the AB2 scheme. The colors of the horizontal lines represent the same variables as the symbols. (b) Resimulation errors compared with the original simulation data,  $\epsilon_{\varphi, \infty}$ . In both plots, the dash-dotted line has a slope of 2.

Beyond  $t + \delta t$ , the resimulation can proceed with AB2 using the original time step  $\delta t$ . For example, the solution at  $t + 2\delta t$  can be computed from information at  $t$  and  $t + \delta t$ ; similarly the solution at  $t + 3\delta t$  can use the information at  $t + \delta t$  and  $t + 2\delta t$  and so on.

The boundary conditions on  $\Gamma$  at the subtime steps can be approximated from temporal interpolation of  $\mathbf{u}_{\text{OS}}^*$  from the original simulation data. For instance, in the example below, the boundary conditions between  $t$  and  $t + \delta t$  are obtained by applying the piecewise cubic Hermite interpolating polynomial on stored boundary conditions (plane data) at  $t - \delta t$ ,  $t$ ,  $t + \delta t$ , and  $t + 2\delta t$ .

For demonstration, we perform a resimulation of the original computation with  $\delta t = 4 \times 10^{-3}$ , starting from  $t = 1$  and advancing the simulation until  $t = 2$ . Resimulations with different numbers of temporal substeps  $k$ , as well as the original AB2 scheme, are compared. Recall that  $k = 1$  is equivalent to the resimulation performing the entire first step with the Euler scheme. In this example, the results from a resimulation with  $k = 1000$  subtime steps are used as the reference data to approximate the “true, exact” values which are unknown. We discard the first few  $\delta t$  to avoid including the pressure jump as seen in the previous examples.

Figure 13(a) shows the maximum relative errors compared with the reference data for  $1 < t < 2$ . The symbols denote the errors between the resimulation and the reference data, which decrease as  $k$  increases. In fact, the errors are proportional to  $k^{-2}$ , or the square of the size of the time substep  $(\delta t/k)^2$ , since the temporal scheme is AB2 in the resimulation. The horizontal lines represent the errors between the original AB2 simulation and the reference data. The errors of the AB2 scheme itself are about  $10^{-5}$ – $10^{-4}$ . Also from Fig. 13(a), it is clear that the errors between the resimulations (symbols) and the original DNS (lines) decrease and then increase as  $k$  increases. However, it should be noted that the differences between the symbols and lines do not equal to the actual errors between the resimulations and the original DNS,  $\epsilon_{\varphi, \infty}$ .

The resimulation errors  $\epsilon_{\varphi, \infty}$ , shown in Fig. 13(b), decrease at a rate of second order in  $k$  before about  $k = 6$ , and then become nearly constant. Although an optimal  $k$  is not observed, the drop of the errors is about two orders of magnitude in the current example. The asymptotic values of  $\epsilon_{\varphi, \infty}$  are also the AB2 errors shown in Fig. 13(a). This example shows that the resimulation errors could decrease by two orders of magnitude with only ten additional time substeps within the first  $\delta t$  from the initial condition, and the minimum errors are bound by those of the AB2 integration scheme in the original simulation.

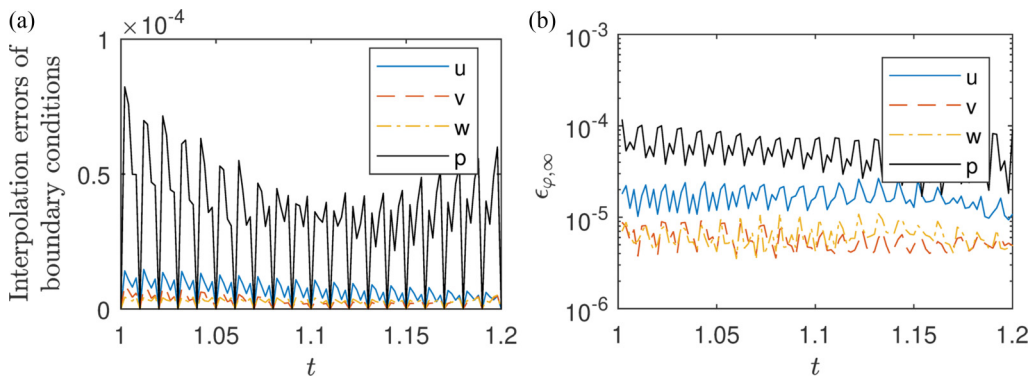


FIG. 14. (a) The interpolation errors of boundary conditions. (b)  $\epsilon_{\varphi,\infty}$  with interpolated boundary conditions. The resimulation is from  $t = 1$  to 2, but only  $t = [1, 1.2]$  is plotted here to more clearly display the oscillations of the errors. The resimulation starts with the AB2 scheme using an extra snapshot provided. The time step size is  $\delta t = 2 \times 10^{-3}$ .

#### D. Temporal subsampling for the boundary conditions

In all previous examples, the resimulations adopted boundary-condition data that were stored at every time step during the original DNS. This may not be necessary or feasible. As mentioned before, the snapshots of the  $1024^3$  isotropic turbulence data set in JHTDB are stored only every ten simulation steps. When data are queried between the two stored time steps, they are obtained with temporal interpolation and the errors are approximately  $10^{-6}$  (we could not determine whether the interpolation errors are lower than  $10^{-6}$ , because the data on JHTDB are stored in single precision). Here we examine the impact of temporal interpolation of temporally subsampled boundary data for resimulation.

We have seen that the resimulation errors are proportional to the errors in the boundary conditions. Thus, if the boundary conditions are stored every few ( $M_{t,bc}$ ) time steps and temporal interpolation is used during resimulation, the errors in the resimulation will be directly proportional to the interpolation errors. Figure 14 shows an example: the time step of the simulation is  $\delta t = 2 \times 10^{-3}$ . The boundary data are stored at every  $M_{t,bc} = 5$  time steps, actually close to the time step requirement based on CFL (based on maximum velocity) equaling to unity. Cubic spline interpolation with three points before and after the query point is used for temporal interpolation. The  $L^\infty$  relative errors of the interpolated boundary-condition fields on the  $\Gamma$  planes are shown in Fig. 14(a). The oscillations of the errors are apparent, vanishing at each of the  $5\delta t$  time instants in which boundary data are known exactly. The resimulation starts at  $t = 1$  using the AB2 scheme with an extra snapshot provided, and runs until  $t = 2$ . As a result, no other errors are introduced in the resimulation, except those due to the temporal interpolation of the boundary conditions. The maximum interpolation errors over time for  $\{u, v, w, p\}$  are  $\{1.47, 1.54, 1.64, 9.66\} \times 10^{-5}$  [Fig. 14(a)]. The resimulation errors  $\epsilon_{\varphi,\infty}$  [Fig. 14(b)] for  $\{u, v, w, p\}$  are  $\{2.66, 2.08, 2.30, 24.2\} \times 10^{-5}$ : all are only slightly higher than the interpolation errors. The oscillations of the resimulation errors are caused by the oscillatory errors of the temporal interpolation of the boundary conditions.

## VI. SUMMARY: RECOMMENDED CHOICES FOR STSR

The previous section has documented separately errors to be expected from various parameter choices for STSR. Here we now combine the various choices that may be expected in an actual implementation of STSR: we use  $\mathbf{u}^*$  on the boundaries stored at every  $M_{t,bc} = 5$  DNS time steps, use  $k = 10$  for the initial temporal subsampling during the first time step of resimulation, use cubic

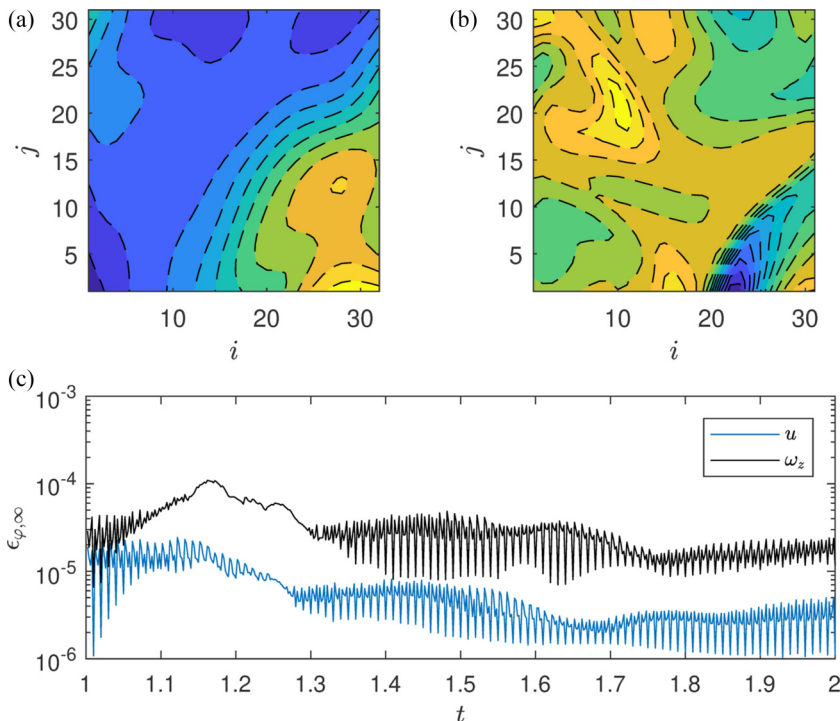


FIG. 15. (a) Contour plot of  $u$  on a randomly selected slice. (b) Contour plot of  $z$ -component vorticity on a randomly selected slice. In (a) and (b), color contours are the original simulation, while the black dashed contour lines are the resimulation. (c)  $L^\infty$  errors of  $u$ - and  $z$ - component vorticity.

polynomial temporal interpolation of the stored  $\mathbf{u}^*$  and  $p$  boundary values to interpolate to the resimulation time step  $\delta t$ , and integrate between  $t = 1$  and 2.

Figure 15 compares two fields at  $t = 2$  from the resimulation to the original simulation: (a)  $u$  velocity and (b)  $z$ -component vorticity  $\omega_z$  (computed using centered finite differencing). The contour lines of resimulation fields and the original ones are on top of each other.

Figure 15(c) shows the corresponding evolution of the  $L^\infty$  errors. The vorticity error is about one order of magnitude higher than the velocity error and is about  $10^{-4}$ . This level of difference between resimulation and original DNS is acceptable and falls within the desired guidelines.

## VII. CONCLUSIONS

In the present paper, we propose an idea of data compression for numerical simulation results of incompressible fluid flow. The entire simulation domain of the original simulation is divided into multiple small subregions by planes. The data in the entire domain are stored, say, at every few hundred or thousand time steps, while data on the dividing planes are stored at every time step, or subsampled every few time steps. Once data at an arbitrary position and time are needed, a resimulation of the small cube region (subdomain) which includes that point is performed. The data stored in the entire domain are used as the initial condition, while the planar data surrounding the subdomain are used as the boundary conditions.

It is found that if the numerical scheme in the resimulation matches the original simulation exactly the resimulation will produce error-free results. On the other hand, any mismatch between the resimulation and the original one can produce significant errors, exceeding the minimum error

levels one would like to enforce for a database that contains spatially and temporally subsampled data.

For example, it was found that resimulation errors are too high when using velocity and pressure differences (or pressure) for the boundary condition. It was shown that the correct velocity boundary conditions for the resimulation should be the intermediate velocity after the projection step: this is because the boundaries of the subdomain are still the internal part of the entire domain of the original simulation.

Another example is that the resimulation should use the same time integration scheme as the original simulation. This poses a challenge if only one snapshot of the initial field is provided: the resimulation must start with an Euler scheme while the original simulation has been advanced with an AB2 scheme. The challenge can be resolved by storing extra snapshots so that the resimulation could start with the AB2 scheme as well, or could be improved using Euler-AB2 integration with several subtime steps to approximate the first AB2 integration in the original simulation. We have shown that the latter approach saves storage space, and can also reduce the resimulation errors by two orders of magnitude with only ten subtime steps added in the first original time step.

The findings also imply that if the original simulation contains source terms in the Navier-Stokes equations, such as in forced isotropic flow, these source terms must also be recorded together with the original simulation and included in the resimulation.

Tests using boundary data with added noise show that resimulation errors remain linearly proportional to the errors in the boundary conditions. This observation helps explain several trends in resimulation errors. Also, it provides a guideline about how much temporal subsampling of the boundary data may be used. The resulting errors in resimulation will be proportional to the errors caused by temporal interpolation on the boundary data. Experiment shows that the resimulation error is similar to the interpolation errors of the boundary conditions. Thus, in a real application, one could carefully control the interval of two stored plane data and achieve further compression of the simulation data.

A sample application combining all of the recommended subsampling parameters and resimulation strategies shows relative maximum error in velocity on the order of  $10^{-5}$  to  $10^{-4}$ , which is acceptable and leads to errors of less than 0.1% in velocity gradients. These levels are acceptable for applications of building numerical turbulence databases like JHTDB. It is worth reiterating that the errors in the numerical experiments performed here did not reveal exponential growth in time, at least not over the tested time horizons. From the viewpoint of data assimilation [20,22–24], synchronization of chaos [21], and nudging [25] of Navier-Stokes turbulence, the present results have implications on the effectiveness of time-evolving boundary conditions to constrain and effectively synchronize or nudge the dynamics. In prior work [20] it was shown that providing the correct large-scale Navier-Stokes dynamics at all wave numbers down to  $\approx 0.2k_\eta$  (i.e., corresponding to grid spacing of  $\approx 15\eta$ ) leads to eventual slaving (synchronization) of the smaller scales, while coarser truncation leads to chaotic divergence of trajectories at the small scales (similar results were obtained later in [21]). Here we show something different: that domains of significantly larger size  $(30\eta)^3$  can still remain slaved to the dynamics at all scales provided the data at the boundaries contain scales down to the smallest viscous scales (DNS resolution). A more systematic analysis, such as testing how large the resimulation subdomain can be made before the boundary information is no longer able to synchronize the dynamics in the core of the subdomain, is beyond the scope of the present paper.

The subsampling and local resimulation technique described in this paper could also be applied on unstructured meshes, as long as the correct information is stored during the original simulation and the resimulation uses exactly the same method as the original simulation, but the compression ratio will be grid-topology dependent. If a spectral method is used in the original simulation (such as in several of the existing JHTDB data sets), using local resimulation with, e.g., finite differencing will lead to significant errors. If the spectral method is used only in one or two directions, like channel flow, good accuracy can be achieved if the resimulation domain consists of all 1D “pencils” or 2D “slabs.” However, if the spectral method is used in all three directions, the present technique

cannot reproduce error-free data unless the resimulation is performed on the entire (large) domain, which is expected to be prohibitive.

We remark that alternative resimulation methods, e.g., based on machine learning tools instead of grid-based CFD methods, could be considered. For instance, one could apply physics informed neural network methods [36] to train an artificial neural network constrained by Navier-Stokes equations to predict field data at desired points and time using similar types of initial and bounding surface data as used in the present method as inputs (see also [37] for a recent example). The present results documenting errors to be expected from Navier-Stokes based resimulation can serve as a useful reference or benchmark to which to compare such alternative methodologies.

Finally, although this paper is focused on turbulence in incompressible flows, extensions of the basic idea and methodological requirements to other fields of computational physics appear possible. Also, other compression tools can be applied on top of the present technique. For example, one can use wavelet methods [38] to further compress the planar and volumetric data.

### ACKNOWLEDGMENTS

The authors acknowledge funding from the National Science Foundation (Grant No. OCE-1633124). Computations were made possible by the Maryland Advanced Research Computing Center and the SciServer platform. Useful discussions and conversations with Prof. A. Szalay, Prof. R. Burns, Prof. G. Eyink, and Dr. C. Lalescu are gratefully appreciated.

### APPENDIX: FAST POISSON SOLVER FOR RESIMULATION

In this Appendix, details about a spectral fast Poisson solver for Eq. (5) used in resimulations are described. Since the resimulation subdomain is in general not periodic, a fast Poisson solver using discrete sine and cosine transforms [33] is implemented.

Consider a one-dimension Poisson equation,

$$\nabla^2 \psi = b, \quad (\text{A1})$$

on a uniform grid  $x_i = ih$  ( $i = 1, \dots, N$ ), where  $h = \Delta x$  is the constant grid spacing. The Poisson equation discretized with second-order central finite differences is

$$\frac{\psi_{i-1} - 2\psi_i + \psi_{i+1}}{h^2} = b_i, \quad i = 1, \dots, m, \quad (\text{A2})$$

and can be represented in Fourier space as

$$\lambda_j \hat{\psi}_j = \hat{b}_j, \quad j = 1, \dots, N, \quad (\text{A3})$$

where  $\lambda = -k'^2$  is the eigenvalue and  $k'$  is the modified wave number. Thus, the Poisson equation can be solved in three steps: (i) calculate  $\hat{b}_j$  from the forward the Fourier, sine, or cosine transform of  $b$ ; (ii) find  $\hat{\psi}_j = \hat{b}_j/\lambda_j$  from Eq. (A3); and (iii) calculate  $\psi$  from the inverse transform of  $\hat{\psi}_j$ . The transforms used in (i) and (iii) and the eigenvalues  $\lambda_j$  depend on the boundary conditions and are listed in Tables II and III. In Table II, ‘‘DFT’’ refers to the discrete Fourier transform, ‘‘DST-II’’

TABLE II. The transforms used in steps 1 and 3 in the fast Poisson solver.

Boundary conditions	Forward	Backward
Periodic ( $x_0 = x_m$ , $x_{m+1} = x_1$ )	DFT	Inverse of DFT
Dirichlet on cell faces ( $x_1 + x_0 = 0$ , $x_{m+1} + x_m = 0$ )	DST-II	Inverse of DST-II
Neumann on cell faces ( $x_1 - x_0 = 0$ , $x_{m+1} - x_m = 0$ )	DCT-II	Inverse of DCT-II



TABLE III. The eigenvalues used in step 2 in the fast Poisson solver.

Boundary conditions	Eigenvalues
Periodic ( $x_0 = x_n, x_{m+1} = x_1$ )	$\lambda_k = -\frac{4}{h^2} \sin^2 \frac{(k-1)\pi}{m}$
Dirichlet on cell faces ( $x_1 + x_0 = 0, x_{m+1} + x_m = 0$ )	$\lambda_k = -\frac{4}{h^2} \sin^2 \frac{k\pi}{2m}$
Neumann on cell faces ( $x_1 - x_0 = 0, x_{m+1} - x_m = 0$ )	$\lambda_k = -\frac{4}{h^2} \sin^2 \frac{(k-1)\pi}{2m}$

refers to type-II discrete sine transform, and ‘‘DCT-II’’ refers to type-II discrete cosine transform. For nonhomogeneous boundary conditions,  $b_1$  and  $b_n$  can be modified in order to absorb the values at the boundaries.

When  $\lambda_1 = 0$ , an additional equation is required; e.g., with the periodic or Neumann boundary conditions in all directions one could simply set  $\hat{\psi}_1 = 0$  leading to a zero-mean solution. It should also be noted that this algorithm gives the least-square solution for the discretized Poisson equation if the compatibility condition  $\sum b_i = 0$  is not satisfied.

The discrete Fourier, sine, and cosine transforms are included in various libraries, including FFTW and FFTPACK. If a DST-II or DCT-II is not implemented, e.g., in the Intel Math Kernel Library (MKL), it can be computed via a DCT-III combined with  $\mathcal{O}(2n)$  pre- and postprocessing.

Extension of the algorithm to three dimensions is straightforward: (i) calculate  $\hat{b}_{j_1 j_2 j_3}$  from the forward transform of  $b$ ; (ii) find  $\hat{\psi}_{j_1 j_2 j_3} = \hat{b}_{j_1 j_2 j_3} / \lambda_{j_1 j_2 j_3}$ , where  $\lambda_{j_1 j_2 j_3} = \lambda_{j_1} + \lambda_{j_2} + \lambda_{j_3}$ ; and (iii) calculate  $\psi$  from the backward transform of  $\hat{\psi}_{j_1 j_2 j_3}$ .

If the grid is nonuniform in only one direction, e.g., in channel or boundary-layer flows, the spectral approach is adopted in all dimensions where the grid is uniform, and a tridiagonal solver is adopted in the direction of grid stretching (see Moin [39, Sec. 6.2.1] for an example). In fact, solving a tridiagonal linear system is faster than Fourier transforms, since the former has a computational cost  $\mathcal{O}(N)$ , which is less than that of fast Fourier transform,  $\mathcal{O}(N \log N)$ .

The current fast Poisson solver is faster in time and saves the memory compared with a Poisson solver implementing a sparse matrix solver. Table IV compares the time spent in solving the discrete Poisson equation using sparse matrix lower-upper (LU) decomposition, FFT, and DST/DCT. When the grid is composed of  $128^3$  points, the LU decomposition requires extensive memory and in our tests using limited resources (as one would like to use during resimulation) it runs out of memory. The solution using DST/DCT requires approximately twice the time of the DFT, and only one-dimensional DST/DCT is available in the majority of numerical libraries. Nevertheless, DST/DCT

TABLE IV. Time spent in solving the discrete Poisson equation with a sparse matrix solver, FFT or DST/DCT. The timing has a resolution of  $10^{-3}$  s, and is averaged over 100 runs. In the LU decomposition method, only the solution phase (i.e., forward and backward substitutions after the LU decomposition) is timed. The hardware is Intel Core i5-7500 (four Cores, 3.4 GHz) and 16-GB memory. The code uses an Intel FORTRAN compiler, Intel MKL, and OPENMP in Windows. The parallelization of the sparse matrix solver and the DFT is implemented in Intel MKL, while that of DST/DCT is implemented by authors using OPENMP. In the  $128^3$  case, the LU decomposition runs out of memory.

Grid points	LU decomposition	DFT	DST/DCT
$32^3$	0.0082 s	$<10^{-3}$ s	$<10^{-3}$ s
$48^3$	0.0357 s	$<10^{-3}$ s	0.0016 s
$64^3$	0.1137 s	0.0019 s	0.0035 s
$96^3$	0.5451 s	0.0052 s	0.0101 s
$128^3$	N/A	0.0109 s	0.0234 s

outperforms the direct solver based on the sparse matrix LU decomposition, and its scalability is superior.

- 
- [1] J. Kim, P. Moin, and R. Moser, Turbulence statistics in fully developed channel flow at low Reynolds number, *J. Fluid Mech.* **177**, 133 (1987).
  - [2] P. Moin and K. Mahesh, Direct numerical simulation: A tool in turbulence research, *Annu. Rev. Fluid Mech.* **30**, 539 (1998).
  - [3] D. Livescu and J. R. Ristorcelli, Variable-density mixing in buoyancy-driven turbulence, *J. Fluid Mech.* **605**, 145 (2008).
  - [4] P. K. Yeung, D. A. Donzis, and K. R. Sreenivasan, Dissipation, enstrophy and pressure statistics in turbulence simulations at high Reynolds numbers, *J. Fluid Mech.* **700**, 5 (2012).
  - [5] I. Bermejo-Moreno, J. Bodart, J. Larsson, B. M. Barney, J. W. Nichols, and S. Jones, Solving the compressible Navier-Stokes equations on up to 1.97 million cores and 4.1 trillion grid points, in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis: SC '13* (ACM, New York, 2013), pp. 1–10.
  - [6] P. K. Yeung, X. M. Zhai, and K. R. Sreenivasan, Extreme events in computational turbulence, *Proc. Natl. Acad. Sci. USA* **112**, 12633 (2015).
  - [7] M. Lee and R. D. Moser, Direct numerical simulation of turbulent channel flow up to  $Re\tau = 5200$ , *J. Fluid Mech.* **774**, 395 (2015).
  - [8] J. Lee and T. A. Zaki, Detection algorithm for turbulent interfaces and large-scale structures in intermittent flows, *Computers Fluids* **175**, 142 (2018).
  - [9] Y. Yamamoto and Y. Tsuji, Numerical evidence of logarithmic regions in channel flow at  $Re\tau = 8000$ , *Phys. Rev. Fluids* **3**, 012602 (2018).
  - [10] J. You and T. A. Zaki, Conditional statistics and flow structures in turbulent boundary layers buffeted by free-stream disturbances, *J. Fluid Mech.* **866**, 526 (2019).
  - [11] J. C. del Alamo and J. Jimenez, Spectra of the very large anisotropic scales in turbulent channels, *Phys. Fluids* **15**, L41 (2003).
  - [12] <http://turbulence.pha.jhu.edu/datasets.aspx>.
  - [13] E. Perlman, R. Burns, Y. Li, and C. Meneveau, Data exploration of turbulence simulations using a database cluster, in *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing: SC '07* (ACM, New York, 2007), p. 1.
  - [14] Y. Li, E. Perlman, M. Wan, Y. Yang, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence, *J. Turbulence* **9**, N31 (2008).
  - [15] H. Yu, K. Kanov, E. Perlman, J. Graham, E. Frederix, R. Burns, A. Szalay, G. Eyink, and C. Meneveau, Studying Lagrangian dynamics of turbulence using on-demand fluid particle tracking in a public turbulence database, *J. Turbulence* **13**, N12 (2012).
  - [16] D. Huffman, A method for the construction of minimum-redundancy codes, *Proc. IREE* **40**, 1098 (1952).
  - [17] J. Ziv and A. Lempel, A universal algorithm for sequential data compression, *IEEE Trans. Inf. Theory* **23**, 337 (1977).
  - [18] ISO, ISO/IEC 11172-3:1993: Information technology: Coding of moving pictures and associated audio for digital storage media at up to about 1,5 Mbit/s, Part 3: Audio (1993), <https://www.iso.org/standard/22412.html>.
  - [19] ISO, ISO/IEC 10918-1:1994: Information technology: Digital compression and coding of continuous-tone still images: Requirements and guidelines (1994), <https://www.iso.org/standard/18902.html>.
  - [20] K. Yoshida, J. Yamaguchi, and Y. Kaneda, Regeneration of Small Eddies by Data Assimilation in Turbulence, *Phys. Rev. Lett.* **94**, 014501 (2005).

- [21] C. C. Lalescu, C. Meneveau, and G. L. Eyink, Synchronization of Chaos in Fully Developed Turbulence, *Phys. Rev. Lett.* **110**, 084102 (2013).
- [22] C. Foias, C. F. Mondaini, and E. S. Titi, A discrete data assimilation scheme for the solutions of the two-dimensional Navier-Stokes equations and their statistics, *SIAM J. Appl. Dyn. Syst.* **15**, 2109 (2016).
- [23] M. Wang, Q. Wang, and T. A. Zaki, Discrete adjoint of fractional-step incompressible Navier-Stokes solver in curvilinear coordinates and application to data assimilation, *J. Comput. Phys.* **396**, 427 (2019).
- [24] J. L. Callahan, K. Maeda, and S. L. Brunton, Robust flow reconstruction from limited measurements via sparse representation, *Phys. Rev. Fluids* **4**, 103907 (2019).
- [25] P. Clark Di Leoni, A. Mazzino, and L. Biferale, Synchronization to Big Data: Nudging the Navier-Stokes Equations for Data Assimilation of Turbulent Flows, *Phys. Rev. X* **10**, 011023 (2020).
- [26] A. Kravchenko and P. Moin, On the effect of numerical errors in large eddy simulations of turbulent flows, *J. Comput. Phys.* **131**, 310 (1997).
- [27] J. van Kan, A second-order accurate pressure-correction scheme for viscous incompressible flow, *SIAM J. Sci. Stat. Comput.* **7**, 870 (1986).
- [28] J. B. Bell, P. Colella, and H. M. Glaz, A second-order projection method for the incompressible Navier-Stokes equations, *J. Comput. Phys.* **85**, 257 (1989).
- [29] L. Nicolaou, S. Jung, and T. Zaki, A robust direct-forcing immersed boundary method with enhanced stability for moving body problems in curvilinear coordinates, *Comput. Fluids* **119**, 101 (2015).
- [30] A. J. Chorin, Numerical solution of the Navier-Stokes equations, *Math. Comput.* **22**, 745 (1968).
- [31] J. Kim and P. Moin, Application of a fractional-step method to incompressible Navier-Stokes equations, *J. Comput. Phys.* **59**, 308 (1985).
- [32] F. H. Harlow and J. E. Welch, Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface, *Phys. Fluids* **8**, 2182 (1965).
- [33] U. Schumann and R. A. Sweet, Fast Fourier transforms for direct solution of Poisson's equation with staggered boundary conditions, *J. Comput. Phys.* **75**, 123 (1988).
- [34] P. M. Gresho and R. L. Sani, On pressure boundary conditions for the incompressible Navier-Stokes equations, *Int. J. Numer. Methods Fluids* **7**, 1111 (1987).
- [35] S. Abdallah and J. Dreyer, Dirichlet and Neumann boundary conditions for the pressure poisson equation of incompressible flow, *Int. J. Numer. Methods Fluids* **8**, 1029 (1988).
- [36] M. Raissi, P. Perdikaris, and G. E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.* **378**, 686 (2019).
- [37] K. Fukami, K. Fukagata, and K. Taira, Super-resolution reconstruction of turbulent flows with machine learning, *J. Fluid Mech.* **870**, 106 (2019).
- [38] K. Schneider and O. V. Vasilyev, Wavelet methods in computational fluid dynamics, *Annu. Rev. Fluid Mech.* **42**, 473 (2010).
- [39] P. Moin, *Fundamentals of Engineering Numerical Analysis*, 2nd ed. (Cambridge University, Cambridge, England, 2010), p. 256.