Documentation for the simulation and analysis package SAINT-VENANT (ver. 1.0)

A. Azzalini[1], M. Caldoro[1], M. Farge[1], G. Perret[1] & B. Protas[2]

July 9, 2004

1) LMD–CNRS, École Normale Supŕieure
24, rue Lhomond, 75231 Paris, Cedex 05, FRANCE

2) Department of Mathematics & Statistics, McMaster University
1280 Main Street West, Hamilton, Ontario, CANADA L8S 4K1

Official website: `http://www.math.mcmaster.ca/~bprotas/Software/sv`

Contents

0 Preamble

This scientific software package is the result of the work of several people over many years and is still under development (see [1] for an early description of the code). Recently we decided to put together a revised version that is now publicly available under the terms of the GNU General Public License[1]. This is the first release of the package and we are aware that there may still be many imperfections, as certain elements of the package are incomplete, whereas other may have not been thoroughly tested. As time permits, we will work to expand the package and make it more reliable. We encourage people to use this code for research and / or study and look forward to receiving any comments, remarks, suggestions for improvement, etc.

The list of people involved at various stages in the development of this code includes (in chronological order): Marie Farge, Alexandre Azzalini, Michele Caldoro, Gaele Perret and Bartosz Protas.

1 Outline of the Problem

The code package SAINT–VENANT consists of three basic units:

- *SV_simul* — which solves numerically the Shallow Water and Navier–Stokes systems,
- *SV_filter* — which performs linear and nonlinear Fourier– and wavelet–based filtering of fields obtained using *SV_simul*,
- *SV_analysis* — which performs the statistical analysis of the fields obtained using *SV_simul* and *SV_filter* (in fact, this unit is still under development and has not been released yet).

Depending on the value of the input parameter `INCOMP`, the code *SV_simul* solves one of the following is two systems of equations:

- when `INCOMP=0` the 2D Shallow Water System is solved (the *compressible* case)

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + (\xi + f)\,\mathbf{k}\times\mathbf{u} + \nabla\left(\Phi + \dfrac{\mathbf{u}^2}{2}\right) - \nu\Delta\mathbf{u} = 0, \\ \dfrac{\partial \Phi}{\partial t} + \nabla\cdot(\Phi\mathbf{u}) = 0, \end{cases} \tag{1}$$

  where $\mathbf{u} = [u, v]$ is the velocity field, $\Phi = gh$ is the geopotential (with g representing the gravitational acceleration and h the height of the water surface), $\xi = \frac{\partial v}{\partial x} - \frac{\partial u}{\partial y}$ is the vorticity, f is the background rotation and ν the kinematic viscosity.

[1] For details see `http://www.gnu.org/copyleft/gpl.html`.

- when `INCOMP=1,2` the 2D Navier–Stokes System[2] is solved (the *incompressible* case); as explained in §2.2, the two cases `INCOMP=1` and `INCOMP=2` correspond to different time–stepping schemes employed

$$\begin{cases} \dfrac{\partial \mathbf{u}}{\partial t} + \xi \mathbf{k} \times \mathbf{u} + \nabla \left(\Phi + \dfrac{\mathbf{u}^2}{2} \right) - \nu \Delta \mathbf{u} = 0, \\ \nabla \cdot \mathbf{u} = 0, \end{cases} \tag{2}$$

  where Φ now has the meaning of the hydrostatic pressure.

The flow domain is a rectangular box $\Omega = [0, L_x] \times [0, L_y]$ with periodic boundary conditions in both directions. It is assumed that $L_x = L_0 = 2\pi$ and $\alpha = \frac{L_x}{L_y}$. As described below, using the volume penalization method, it is possible to incorporate solid boundaries in the flow domain. A version of the code that allows for the presence of solid boundaries (using the volume penalization method [2]) is also available.

The specific flow configuration is determined based on the values of the input parameters `INCOMP` (see above) and `FLOW_CASE`. The latter can assume the following values:

- `FLOW_CASE=0` — homogeneous and isotropic case without solid boundaries,
- `FLOW_CASE=1` — the coordinate system coincides with the center of a translating obstacle,
- `FLOW_CASE=2` — the coordinate system coincides with the obstacle which remains motionless; upstream current in geostrophic balance is added,
- `FLOW_CASE=3` — the coordinate system is attached to the container, rather than to the translating obstacle.

The presence and the shape of the solid body is determined by the parameter `PENALIZATION` which can assume the following values:

- `PENALIZATION=0` — there is no solid body,
- `PENALIZATION=1` — the solid body is a cylinder with the diameter D=COEF_DIAM_LO*L0/α, where `COEF_DIAM_LO` has to be specified in the parameter file (see §5),
- `PENALIZATION=2` — the solid body is a smeared cylinder, i.e., the mask of the body is equal to one for $r \leq (R - \delta)$ and then decreases exponentially for $(R - \delta) < r \leq R$,
- `PENALIZATION=3` — the solid body is a square with the dimension a =COEF_DIAM_LO*L0/(2*α),
- `PENALIZATION=4` — the mask for the solid body is read from a file called *Mask_Xi*,
- `PENALIZATION=5` — the solid body is an ellipse with the minor axis a =COEF_DIAM_LO*L0/(2*α) and the major axis b =1.8*COEF_DIAM_LO*L0,

[2] When working with spatially–periodic domains it is customary to solve the 2D Navier–Stokes system in the vorticity formulation. Here, however, we choose to use the primitive (i.e., velocity–pressure) formulation so as to make this case consistent with the Shallow Water version developed initially and also to facilitate subsequent extension to three dimensions.

- `PENALIZATION=6` — used for flows in a cylindrical tank; the equations are then penalized at the perimeter of the tank; the diameter of the tank is D=COEF_DIAM_LO*L0/α.

For all those case, the center of the obstacle is located at $[L_x/(4\alpha), L_y/2]$ when `FLOW_CASE < 2`. When `FLOW_CASE=3`, the initial position of the obstacle is $[L_x - L_x/(4\alpha), L_y/2]$. It is also possible to include a sponge region at the end of the domain (when `SPONGE=1`) and / or lateral boundaries in the streamwise direction (when `WALLS=1`).

All the input parameters are specified in the parameter file (see §5).

The code *SV_filter* first calculates the wavelet (or Fourier) transform of a given field (for instance, vorticity) and then splits this field into the coherent (or large scale) and incoherent (or small scale) part. The threshold for splitting is either fixed, or adaptively determined using various criteria (see §2.5 for details). The code can also calculate an approximation of the Besov norm of a given field based on its discrete wavelet representation as in the following 1D example

$$\left\| f \right\|^r_{\mathcal{B}_{p,q}} = |\alpha_0| + \left\{ \sum_{j=0}^{N} \left[2^{j(s+\frac{1}{2}-\frac{1}{p})} \left(\sum_k |\beta_{jk}|^p \right)^{1/p} \right]^q \right\}^{1/q}, \tag{3}$$

where $\alpha_k = (f, \Phi_k)$ and $\beta_{jk} = (f, \Psi_{jk})$ are scaling function and wavelet coefficients.

The code *SV_analysis* performs statistical analyses of a given field calculating quantities such as energy spectra, probability distribution functions (PDFs) of the field, its derivatives and its increments.

2 Numerical Method

2.1 Spatial Discretization

Systems (1) and (2) are solved using a standard pseudo–spectral method [3] in a periodic domain with $SIZE\_I \times SIZE\_J$ grid points. The numerical resolution in the two directions X and Y does not have to be the same. When it is different, all the variables are nondimensionalized with respect to L_x ($L_y = \alpha L_x$, where $\alpha = \frac{SIZE\_J}{SIZE\_J}$). Regardless of the actual resolution in X and Y, the grid in physical space remains isotropic (i.e., $\Delta x = \Delta y$). As a result, the grid in Fourier space becomes anisotropic (i.e., $\Delta k_y = \alpha \Delta k_x$). The fields $\hat{u}$, $\hat{v}$ and $\hat{\Phi}$ (hats denote 2D Fourier transforms) are represented using $\frac{1}{2}SIZE\_I$ Fourier modes in the X direction and $\frac{1}{2}SIZE\_J$ Fourier modes in the Y direction. The parameters $SIZE\_I$ and $SIZE\_J$ are specified in the *Makefile* where they are used by the preprocessor. Fourier transforms are performed using FFTW routines (`www.fftw.org`). For reasons of consistency with the CRAY SCILIB routines, the FFTW routines are called through wrappers provided in the library *libjmfftw.a*. When `IDALIAS=1`, dealiasing is performed by setting to zero the Fourier coefficients corresponding to the wavenumbers k_x and k_y such that $k_x \geq k_{max}$ or $k_y \geq k_{max}$, where $k_{max} = \min(\frac{SIZE\_I}{3}, \frac{SIZE\_J}{3})$ (we use here the "3/2" rule: $\frac{SIZE\_I}{3} = \frac{2}{3}\frac{SIZE\_I}{2}$). When `IDALIAS=2`, dealiasing is performed by setting to zero the Fourier coefficients corresponding to the wavenumbers k_x and k_y such that $\sqrt{k_x^2 + k_y^2} \geq k_{max}$. Note that in the latter case dealiasing is performed isotropically in Fourier space and is therefore more aggressive than required. When `IDALIAS=0`, dealiasing is not performed.

2.2 Temporal Discretization

In the compressible case (`INCOMP=0`) and in the incompressible case with `INCOMP=1`, time integration is performed by combining exact integration of the viscous and penalization terms with the leapfrog scheme on all the remaining terms (represented by $\mathcal{A}(y)$ in the expressions below)

$$\hat{y}_k^{n+1} = \hat{y}_k^{n-1} e^{-2\nu k^2 \Delta t} + 2\Delta t\, \hat{\mathcal{A}}_k^n(\hat{y})\, e^{-\nu k^2 \Delta t}. \tag{4}$$

In the incompressible case with `INCOMP=2` the leapfrog scheme is replaced with the second–order accurate Adams–Bashforth scheme

$$\hat{y}_k^{n+1} = \hat{y}_k^n e^{-\nu k^2 \Delta t} + \frac{1}{2}\Delta t \left[3\hat{\mathcal{A}}_k^n(\hat{y}) - \hat{\mathcal{A}}_k^{n-1}(\hat{y})\right] e^{-\nu k^2 \frac{\Delta t}{2}}. \tag{5}$$

Here $\hat{y}_k^n$ represents the $k-th$ Fourier component at time $n\Delta t$ of the advanced variable ($\hat{u}$, $\hat{v}$ and $\hat{\Phi}$ in the compressible case, and $\hat{u}$ and $\hat{v}$ in the incompressible case) and Δt is the time step selected to satisfy the CFL condition. When the leapfrog scheme is used, even and odd time steps are resynchronized whenever the relative difference of the kinetic energy or enstrophy (potential enstrophy when `INCOMP=0`) between two consecutive steps exceeds `ECARLF` percent. This is followed by an explicit Euler time step. When initializing the simulation from scratch, time integration is also begun with a single explicit Euler step. For simplicity, an explicit Euler time step is performed by calling the Adams–Bashforth subroutine with $\hat{\mathcal{A}}_k^{n-1}(\hat{y}) = \hat{\mathcal{A}}_k^n(\hat{y})$. A single time step in both compressible and incompressible cases necessitates 12 Fourier transforms.

2.3 Pressure Correction

In the Navier–Stokes case incompressibility is enforced using the fractional step algorithm with pressure correction [4]. We treat the velocity fields obtained from (4) or (5) as intermediate fields (marked with a tilde) and correct them as follows

$$\mathbf{u}^{n+1} = \tilde{\mathbf{u}}^{n+1} - \nabla\varphi, \tag{6}$$

where the *correction* field φ is obtained as

$$\Delta\varphi = \nabla \cdot \tilde{\mathbf{u}}^{n+1}. \tag{7}$$

The pressure is obtained by solving the 2D Poisson equation[3]

$$-\Delta\Phi = 2\left(\frac{\partial u}{\partial x}\right)^2 + 2\left(\frac{\partial u}{\partial y}\right)\left(\frac{\partial v}{\partial x}\right). \tag{8}$$

2.4 Enforcement of Boundary Conditions via Volume Penalization

The volume penalization consists in enforcing the boundary conditions on solid boundaries by adding a penalty term to the governing equation (1) or (2) (cf. [2]). In the compressible (`INCOMP=0`) and incompressible case (`INCOMP=1`) the penalty term is integrated exactly in physical space, i.e., in expressions (4) and (5) the terms $\hat{y}_k^{n-1}$ and $\hat{\mathcal{A}}_k^n$ actually become $\widehat{y_k^{n-1} e^{-2\frac{\chi}{\varepsilon}\Delta t}}$ and $\widehat{\mathcal{A}_k^n e^{-\frac{\chi}{\varepsilon}\Delta t}}$ where χ is the mask of the obstacle and ε is a small parameter. For reasons of stability, ε cannot be smaller than Δt and in the code has been selected as $2\Delta t$.

[3] In fact, in the pressure correction algorithm an approximation to pressure can be obtained based on the function φ. This is however not implemented yet.

2.5 Wavelet Transform and Adaptive Filtering

The Discrete Wavelet Transform is calculated using the function `wxfrm_dand` from the library WVLT — The UBC Imager Wavelet Package (Release 3.0). The wavelet filter must be chosen from the following list: *AdelsonSimoncelliHingorani*, *AntoniniBarlaudMathieuDaubechies_4_4*, *BattleLemarie*, *BurtAdelson*, *Coifman_12*, *Daubechies_4*, *Daubechies_6*, *Daubechies_8*, *Daubechies_10*, *Daubechies_12*, *Daubechies_20*, *Haar*, *Pseudocoiflet_4_4*, *Spline_2_2*, *Spline_2_4*, *Spline_3_3*, *Spline_3_5*, *Spline_3_7*. In the case of Fourier transform the splitting threshold is fixed, whereas in the case of wavelet transform the splitting threshold is selected adaptively using either the MAD algorithm[4] [5], or an iterative recursive algorithm [6].

3 Organization of the Codes

3.1 Code *SV_simul*

The code *SV_simul* is made up of three source files: *modules.F90*, *source.F90* and *MovieOut.c*. All the important data structures are contained in the three modules: `database`, `parameters` and `diagnostics` (file *modules.F90*). The discretized fields are stored in the following core arrays:

- `PU`, `PV` and `PP` (data at time level $n+1$),
- `PUA`, `PVA` and `PPA` (data at time level n),
- `PUAA`, `PVAA` and `PPAA` (data at time level $n-1$),
- `PO` (vorticity),
- `PF` (streamfunction),
- `WORK` (double size auxiliary array),
- `MASK` (mask for dealiasing),
- `DISSIP` (discretized dissipation operator).

Owing to a legacy part of the code[5], fields in Fourier space are stored as (real only) coefficients of the sine and cosine transforms interwoven with one another. The source file *source.F90* contains the main body of the program and definitions of all the functions and subroutines. For ease of navigation, all of the most important blocks of the main code as well as function and subroutine definitions are numbered and marked with lines beginning with the sign "$$" (one can take advantage of that, e.g., using the command *List Matching Lines* in the editor *XEmacs 21.4+*). Calls to FFTW subroutines are made through wrapper available in the library *libjmfftw.a*. The file *MovieOut.c* contains routines necessary for graphical output.

[4] As of today, this option, although implemented in the code, has not been activated yet.

[5] This is related to the fact the implementation of the complex arithmetic in early FORTRAN compilers, notably on CRAY machines, was very ineffi cient as compared to the real arithmetic

3.2 Code *SV_filter*

The structure of the code *SV_filter* is very simple — the source file *SV_filter.c* contains the body of the program, whereas the algorithms for threshold determination are implemented in separate files *RecurFilter.c* and *MedianWltFilter.c*.

3.3 Code *SV_analysis*

Still under development.

4 Installation

The tarball should be placed in a chosen directory, e.g., *$(HOME)/Turb2D/*. Upon untarring the archive, the following subdirectories will be created:

- *Source* — containing the *Makefile* as well as all the source files (i.e., *.F90* and.*c*),
- *lib* — containing the libraries *libfftw.a*, *librfftw.a*, *libmpege.a*, *libjmfftw.a* and *libwvlt.a*,
- *bin* — where the executables will be placed after compilation,
- *wvlt* — containing the source files for the library WVLT — The UBC Imager Wavelet Package (Release 3.0),
- *dweezil* — containing the source files necessary to build the code which handles the graphical output,
- *Documentation* — containing LATEX and plain text files documenting the code, including the present file.
- *Text_case* — containing all the data and parameter files required to run the test case (see §5.1)

All of these settings can be modified by editing *Makefile*. All the files containing the source code for *SV_simul*, *SV_filter* and *SV_analysis* can be archived by executing the command *make gohome*. Execution of the command *make tarball* creates an archive containing the complete distribution, including all libraries (source code and binary files), sample parameter files, documentation and the test case. It is required for compilation and linking that a FORTRAN 90/95 and a C compiler be available on the system. As the C compiler, *gcc* is used by default. All the settings regarding compilers and linkers should be adjusted in *Makefile*. Below we describe the steps required for compilation of the different executables.

4.1 Code *SV_simul*

The executable can be build simply by executing the command *make* in the directory *$(HOME)/Turb2D/Source*. The spatial resolution is hardwired in the code by specifying the values of the variables `SIZE_I` and `SIZE_J` in *Makefile* (these settings can be overridden by assigning new values to these variables on the command line following the invocation of *make*). Compiler optimization options as well should be adjusted in *Makefile* to correspond to the specific compilers used. OpenMP–based parallelization is activated by issuing the following command *make PARALLEL=openmp*. The

following libraries ought to be available in *$(HOME)/Turb2D/lib* (the libraries provided in the tarball are for LINUX with *glibc 2.2+* and will likely have to be rebuilt for different architectures):

- *libfftw.a*, *librfftw.a* and *libjmfftw.a* — the first two of which belong to the package FFTW and can also reside in some standard directory (e.g., */usr/lib*); the library *libjmfftw.a* can be built by executing *make jmfftw* (the option *PARALLEL=openmp* should be added to enable parallelization),
- *libmpege.a* required for generation of the graphical (MPG) output; generation of the graphical output is activated by specifying the preprocessor flag *MPG* in *Makefile*.

The executable *dweezil* can be built by changing the the directory *$(HOME)/Turb2D/dweezil* and then executing the following series commands: *make clean; make distclean; make; make install*, which creates the executable and puts it in the directory *$(HOME)/bin* where it can be accessed by the simulation code.

4.2 Code *SV_filter*

The code *SV_filter* is built by executing the command *make SV_filter*. It is necessary that the library *libwvlt.a* be available in the directory *$(HOME)/Turb2D/lib*. This library can be built by executing the command *make OSREL_MAJOR=1 ARCH=LINUX* in the directory *$(HOME)/Turb2D/wvlt*. Note that the parameters of the *make* command will likely have to be adjusted when building the library on a system different than LINUX.

4.3 Code *SV_analysis*

Still under development.

5 Preprocessing, Execution and Postprocessing

5.1 Code *SV_simul*

When the code is launched, the parameter file needs to be piped through the standard input stream (`UNIT=5`), i.e. the following command has to be issued in the execution directory *./a.out_512x512 ¡ params_simul.dat*. The meaning of all the parameters in the parameter file is explained in the sample file provided in *$(HOME)/Turb2D/Source*. As regards the initial condition, two options are available:

- when `IINIT=1` the initial condition is created from scratch in accordance with the settings specified in the parameter file; in the currently available cases the initial fields have Fourier coefficients with a prescribed spectral slope and random phases,
- when `IINIT=0,2` initialization is made with a restart file obtained using a different run with similar parameters (e.g., files *field_final*, *field_0000XYZ*); the restart file must be renamed *fort.1*; when `IINIT=2` the inertio–gravity modes are removed from the restart field (compressible case only).

Possible forms of output from the code depend on the value of the various parameters specified in the parameter file *params_simul.dat* and include:

- unformatted binary files with the fields $\hat{u}$, $\hat{v}$ and $\hat{\Phi}$ at the time steps n and $n+1$ saved every `NSAVE` steps (when `ISAVE=1`); these fields are saved in spectral representation and can be used to reinitialize a new run (see above),

- unformatted binary files containing any of the fields {*vorticity, potential vorticity, pressure, divergence, streamfunction, horizontal velocity, vertical velocity, velocity modulus, Bernoulli potential*} in the physical space representation saved every `NSXXXXX` time steps when `ISXXXXX=1`, where "`XXXXX`" is to be selected from the set {`VORT`, `VPOT`, `PRESS`, `DIV`, `COUR`, `U`, `V`, `MODV` and `BERN`},

- ASCII text files with various run–time diagnostics computed every `NINV` time steps; if `VERBOSE>0` this information is also output to the screen; histories of the following quantities are saved:

  – incompressible case `INCOMP=1`:

    * time t,
    * energy $E = \|\mathbf{u}\|_{L_2}$,
    * enstrophy $Z = \|\omega\|_{L_2}$,
    * palinstrophy $P = \|\boldsymbol{\nabla}\omega\|_{L_2}$,
    * total divergence $D = \|\boldsymbol{\nabla}\cdot\mathbf{u}\|_{L_1}$,
    * peak velocities $\max_\Omega |\mathbf{u}|$, $\min_\Omega u$, $\max_\Omega u$, $\min_\Omega v$ and $\max_\Omega v$,
    * peak pressures $\min_\Omega \Phi$ and $\max_\Omega \Phi$,
    * stability parameter $\kappa_{CFL} = \pi \cdot \Delta t \cdot k_{max} \cdot \max_\Omega |\mathbf{u}|$,
    * kinematic viscosity ν,

  – compressible case `INCOMP=0`:

    * time t,
    * total energy $E = E_k + E_p$,
    * kinetic energy $E_k = \int_\Omega \mathbf{u}^2 \Phi \, d\Omega$,
    * potential energy $E_p = \|\Phi\|_{L_2}$,
    * linear energy $E_l = \Phi_0 \int_\Omega \mathbf{u}^2 \, d\Omega + E_p$, where $\Phi_0 = \int_\Omega \Phi \, d\Omega$,
    * total enstrophy $Z = \|\omega - Co\|_{L_2}$, where Co is the Coriolis parameter,
    * potential enstrophy $Z_p = \int_\Omega \frac{\omega^2}{\Phi} \, d\Omega$,
    * linear enstrophy $Z_l = \|\omega - Co - \Phi \frac{Co}{\Phi_0}\|_{L_2}$
    * palinstrophy $P = \|\boldsymbol{\nabla}\omega\|_{L_2}$,
    * total divergence $D = \|\boldsymbol{\nabla}\cdot\mathbf{u}\|_{L_1}$,
    * peak velocities $\max_\Omega |\mathbf{u}|$, $\min_\Omega u$, $\max_\Omega u$, $\min_\Omega v$ and $\max_\Omega v$,
    * peak pressures $\min_\Omega \Phi$ and $\max_\Omega \Phi$,
    * stability parameter $\kappa_{CFL} = \pi \cdot \Delta t \cdot k_{max} \cdot \max_\Omega |\mathbf{u}|$
    * kinematic viscosity ν,
    * Mach number M
    * Rossby number Ro

  As regards the peak values, the maxima and minima are taken over the whole flow domain Ω.

- video MPG files with animations of any of the fields {*vorticity, potential vorticity, pressure, divergence, streamfunction, horizontal velocity, vertical velocity, velocity modulus, Bernoulli potential*} in the physical space representation saved every `NMPGXXXXX` time steps when `IMPGXXXXX=1` and `VERBOSE>0`, where "`XXXXX`" is to be selected from the set {`VORT`, `VPOT`, `PRESS`, `DIV`, `COUR`, `U`, `V`, `MODV` and `BERN`}; this operation is in fact performed by the code *dweezil* which is called from within the simulation code and must therefore reside in a directory listed in the `PATH` environmental variable (e.g., *$(HOME)/bin*).

The code is also provided with a test case which consists of the following files (all available in the directory *$(HOME)/Turb2D/Test_case*):

- parameter file *params.dat*,
- binary (small endian) restart file *fort.1*,
- diagnostics output file *IRVOR0.dat*,

which can be used for consistency checks. The test case consists in the solution of an incompressible flow problem, thus the results concerning the total divergence D (fifth column in the file *IRVOR0.dat*) should be on the order of the machine epsilon and may have different numerical values on different machines.

5.2 Code *SV_filter*

The filtering code can be executed as *./SV_filter field ¡ params_filter.dat*, where *field* is the field to be analyzed (either ASCII or binary), whereas the file *params_filter.dat* specifies all the parameters regarding the input file *field* (i.e., format, dimension, size) and the operations to be performed. See the sample parameter file provided in *$(HOME)/Turb2D/Source*) for a detailed description. The file *field* can contain any field represented in physical space and obtained, for instance, using[6] *./a.out_512x512*. The code *SV_filter* can return files with coefficients of the discrete wavelet transform (DWT), or with fields corresponding to the coherent and incoherent part of the original field. When the filtering threshold is determined using the recursive algorithm, the results for the consecutive iterations are returned in *iters.dat* and also on the screen when the verbosity level is higher than 1. When the Discrete Wavelet Transform is computed only (no filtering), the Besov norms (cf. (3)) are also calculated for a range of parameters specified in the parameter file. Sample parameter files *params_filter1.dat* and *params_filter2.dat* (for the 1D and 2D cases, respectively) containing explanations for all input parameters are provided in the directory *$(HOME)/Turb2D/Source*.

5.3 Code *SV_analysis*

Still under development.

6 Acknowledgments

Financial support from SPI–CNRS is gratefully acknowledged.

[6] For example, assume that the file *vorticity* consists of 10 instantaneous fields at resolution 512^2 in double precision. It will then have the size of 20971520 bytes ($512^2 \times 8 \times 10 = 20971520$). Instantaneous fields needed for *SV_filter* can be extracted from this file using the UNIX/LINUX command *split -b 2097152 vorticity*.

A *Makefile* options

The following is a summary of options available for the various *Makefiles* provided with the package:

- in *$(HOME)/Turb2D/Source*)
  - *make* — builds the simulation executable in accordance with the parameters specified in *Makefile*; note that these settings can be overridden by specifying them explicitly on the command line, e.g., *make SIZE_I=1024 SIZE_J=1024*,
  - *make SV_filter* — builds the filtering executable,
  - *make jmfftw* — builds the wrapper for the FFTW calls,
  - *make gohome* — creates an archive with the source files,
  - *make tarball* — creates an archive with a complete distribution of the package, including libraries, etc.,
- in *$(HOME)/Turb2D/wvlt*)
  - *make OSREL_MAJOR=1 ARCH=LINUX* — builds the discrete wavelet transform library *libwvlt.a*
- in *$(HOME)/Turb2D/dweezil*)
  - *make* — builds the executable *dweezil* necessary to process the video (MPG) output.

Apart from the above, standard options such as *make clean* and *make distclean* are also available.

References

[1] Marie Farge, “Dynamique non lineaire des ondes et des tourbillons dans les équations de Saint–Venant”, *Doctorat des Mathématiques*, Universite Paris VI, (1987).

[2] N. K.-R. Kevlahan and J. -M. Ghidaglia, “Computation of turbulent flow past an array of cylinders using a spectral method with Brinkman penalization”, *Eur. J. Mech. B* **20**, 333-350, (2001).

[3] C. Canuto, M. Hussaini, A. Quarteroni and T. Zang, “Spectral Methods in Fluid Dynamics”, Springer Verlag, (1990).

[4] J. Kim and P. Moin, “Application of a fractional step method to incompressible Navier–Stokes equation”, *Journal of Computational Physics* **59**, 308-323, (1985).

[5] S. Mallat, “A wavelet tour of signal processing”, Academic Press, (1998).

[6] A. Azzalini, M. Farge and K. Schneider, “Nonlinear wavelet thresholding: a recursive method to determine the optimal denoising threshold” (submitted), (2003).