

## Large-Eddy Simulation (LES)

**Four parts:**

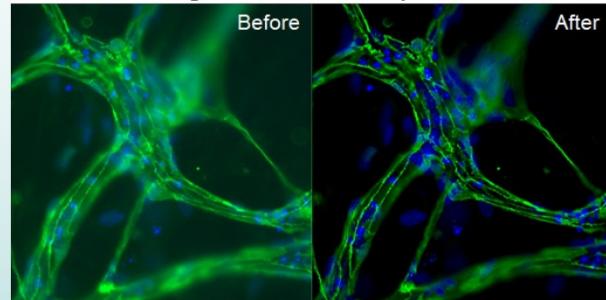
1. Filtering operation
2. Filtered equations
3. Closure: Subfilter modeling (modeling error)
4. Numerics (numerical error)

## VI.1 Filtering

### Applications

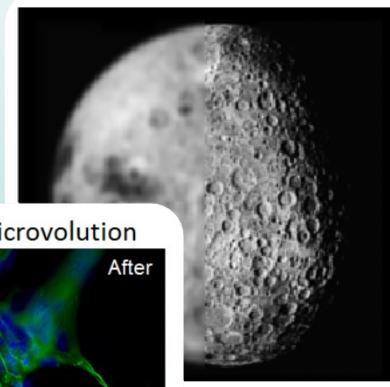
- Image processing
  - deconvolution

Etaluma Image Deconvolved by Microvolution



Cells in 3D channel of microfluidic chip wide-field imaged by Etaluma LS720 Microscope

Chap 6



By E. Amani

## VI.1 Filtering

### Applications

- Image processing
  - Deconvolution
  - ...

Operation	Kernel $\omega$	Image result $g(x,y)$
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Ridge detection	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur $3 \times 3$ (approximation)	$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Chap 6

By E. Amani

## VI.1 Filtering

### Applications

- Image processing
- Signal processing
- Data processing
- ...
- Large Eddy Simulation (LES)

Chap 6

By E. Amani

## VI.1 Filtering

### General definitions and properties of filters

- Idea:
  - Attenuating the **high-frequency** part of the solution
  - Leaving the **low-frequency** part (nearly) unchanged
  - Can be captured with a coarser grid
- A large group of filters: **Convolution filters**

$$\bar{\vec{U}}(\vec{x}, t) = \int_{\mathcal{D}} G(\vec{r}, \vec{x}) \vec{U}(\vec{x} - \vec{r}, t) d\vec{r} = \int_{\mathcal{D}} G(\vec{x} - \vec{r}, \vec{x}) \vec{U}(\vec{r}, t) d\vec{r} \quad (6.1)$$

Domain                  Filter function  
                             (kernel)

- If  $G = \delta(\vec{r}) \rightarrow \bar{\vec{U}} = \vec{U}$

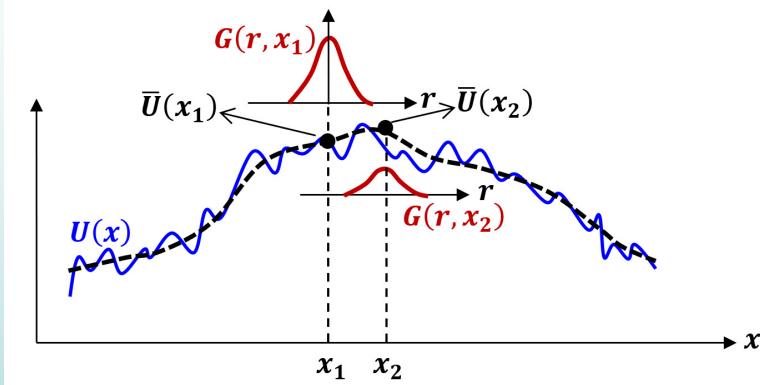
Chap 6

By E. Amani

## VI.1 Filtering

### Convolution filters

$$\bar{U}(\vec{x}, t) = \int_{\mathcal{D}} G(\vec{r}, \vec{x}) \vec{U}(\vec{x} - \vec{r}, t) d\vec{r}$$



▲ Representation of the filter action, Eq. (6.1), in a 1D case.

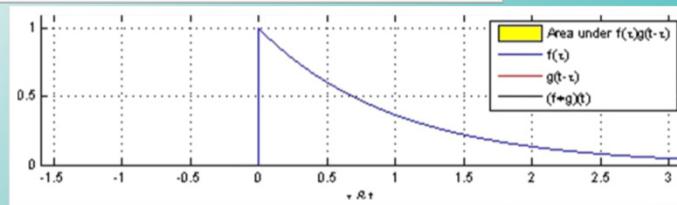
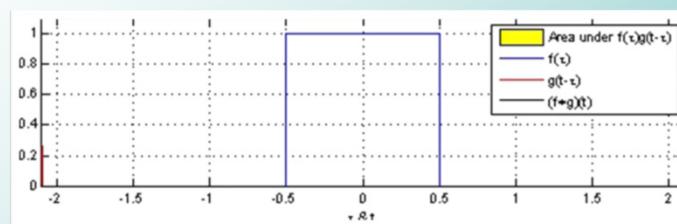
Chap 6

By E. Amani

## VI.1 Filtering

### Convolution filters

▼ Representation of the filter action, Eq. (6.1), in a 1D case.



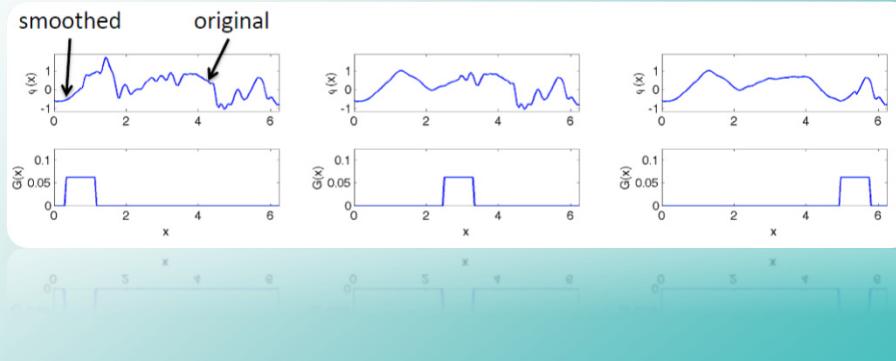
Chap 6

By E. Amani

## VI.1 Filtering

### Convolution filters

▼ Representation of the filter action, Eq. (6.1), in a 1D case.



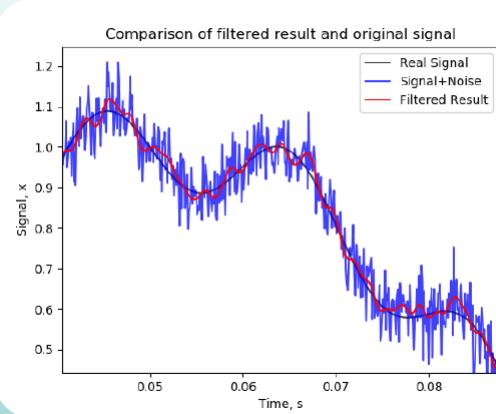
Chap 6

By E. Amani

## VI.1 Filtering

### Convolution filters

▼ Representation of the filter action, Eq. (6.1), in a 1D case.



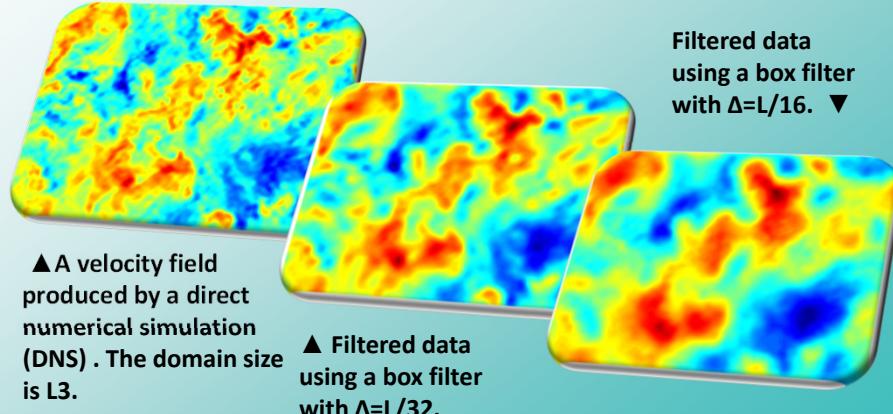
Chap 6

By E. Amani

## VI.1 Filtering

### Convolution filters

▼ Representation of the filter action, Eq. (6.1), in a 3D case of homogeneous decaying turbulence. (2D views).



## VI.1 Filtering

### General definitions and properties of filters

- The normalization condition:

$$\int_{\mathcal{D}} G(\vec{r}, \vec{x}) d\vec{r} = 1; \text{ for each } \vec{x} \quad (6.2)$$

- Exercise: For a simple case of a uniform function  $\vec{U}(\vec{x}, t) = U_0 = cte$ , it is desirable that  $\vec{\bar{U}} = U_0$ . Derive the criterion on the filter kernel which guarantees this property.

- Consequently:

$$\vec{U} = \vec{\bar{U}} + \vec{U}' \quad (6.3) \longrightarrow \vec{U}' = \vec{U} - \vec{\bar{U}} \quad (6.4)$$

Filtered (resolved)      Subfilter (residual)

- Note that (in general):  $\vec{U}' \neq 0, \vec{\bar{U}}' \neq \vec{\bar{U}}$  (6.5)

## VI.1 Filtering

### General definitions and properties of filters

- Filtering properties:

$$\overline{\left( \frac{\partial \vec{U}}{\partial t} \right)} = \frac{\partial \vec{\bar{U}}}{\partial t} \quad (6.6)$$

$$\overline{\langle \vec{U} \rangle} = \langle \vec{\bar{U}} \rangle \quad (6.7)$$

$$\overline{\left( \frac{\partial U_i}{\partial x_j} \right)} \neq \frac{\partial \bar{U}_i}{\partial x_j} \quad (6.8)$$

- Homogeneous filters: The same kernel over all domain

$$G(\vec{r}, \vec{x}) = G(\vec{r}) \quad (6.9)$$

- Homogeneous + spherically symmetric: Isotropic filter

$$G(\vec{r}) = G(r); r = |\vec{r}| \quad (6.10)$$

Chap 6

By E. Amani

## VI.1 Filtering

### General definitions and properties of filters

- Exercise: show that for homogeneous filters

$$\overline{\left( \frac{\partial U_i}{\partial x_j} \right)} = \frac{\partial \bar{U}_i}{\partial x_j} \quad (6.11)$$

- Note: In the next parts, this property may also be used for non-homogenous filters. The arising error is considered as a part of simulation error.

Chap 6

By E. Amani

## VI.1 Filtering

### Filter types

- The box (top-hat) filter:

- 1D box filter

$$G(r) = \frac{1}{\Delta} H\left(\frac{\Delta}{2} - |r|\right) \quad (6.15)$$

**Filter width**

- 3D box filter

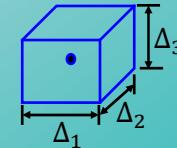
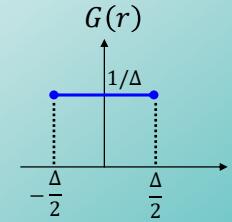
$$G(\vec{r}) = \prod_{i=1}^3 \frac{1}{\Delta_{(i)}} H\left(\frac{1}{2}\Delta_{(i)} - |r_{(i)}|\right) \quad (6.14)$$

$$\bar{U}(\vec{x}, t) = \frac{1}{\Delta_1 \Delta_2 \Delta_3} \iiint_B \bar{U}(\vec{x}', t) d\vec{x}'$$

- Equivalent to the volume averaging in the cubic box

Chap 6

By E. Amani



## VI.1 Filtering

### Filter types

- The box (top-hat) filter:

- 1D box filter

$$G(r) = \frac{1}{\Delta} H\left(\frac{\Delta}{2} - |r|\right) \quad (6.15)$$

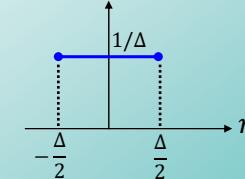
- Exercise: For calculating  $\bar{U}$ , the box filter, Eq. (6.15), uses the values of  $U(x)$  at all nodes within  $-\frac{\Delta}{2} \leq x \leq \frac{\Delta}{2}$ . For calculating  $\bar{U}$  which values of  $U(x)$  are used by the box filter? Are  $\bar{U}$  and  $\bar{U}$  identical?

- The 1D box filter, Eq. (6.15) is symmetric (isotropic). But the 3D filter Eq. (6.14) is not.

- A 3D isotropic box filter variant:

$$G(\vec{r}) = G(r) = \frac{6}{\pi \Delta^3} H\left(\frac{\Delta}{2} - |r|\right) \quad (6.16)$$

G(r)



Chap 6

By E. Amani

## VI.1 Filtering

### Filter types

Name	Filter function	Transfer function
General	$G(r)$	$\hat{G}(\kappa) \equiv \int_{-\infty}^{\infty} e^{ikr} G(r) dr$
Box	$\frac{1}{\Delta} H(\frac{1}{2}\Delta -  r )$	$\frac{\sin(\frac{1}{2}\kappa\Delta)}{\frac{1}{2}\kappa\Delta}$
Gaussian	$\left(\frac{6}{\pi\Delta^2}\right)^{1/2} \exp\left(-\frac{6r^2}{\Delta^2}\right)$	$\exp\left(-\frac{\kappa^2\Delta^2}{24}\right)$
Sharp spectral	$\frac{\sin(\pi r/\Delta)}{\pi r}$	$H(\kappa_c -  \kappa ),$ $\kappa_c \equiv \pi/\Delta$
Cauchy	$\frac{a}{\pi\Delta[(r/\Delta)^2 + a^2]}, \quad a = \frac{\pi}{24}$	$\exp(-a\Delta \kappa )$
Pao		$\exp\left(-\frac{\pi^{2/3}}{24}(\Delta \kappa )^{4/3}\right)$

◀ Table 13.2 [1],  
filters formula in  
physical space

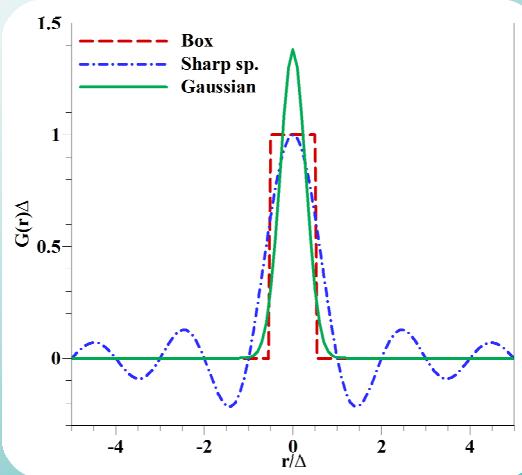
Chap 6

By E. Amani

## VI.1 Filtering

### Filter types

Representation of filters in physical space. Solid line: Gaussian filter, dashed line: box filter, dash-dotted: sharp spectral filter.►



Chap 6

By E. Amani

## VI.1 Filtering

### Implicit filtering – numerical diffusion

→ Lecture Notes: 6.1.3

Chap 6

By E. Amani

## VI.1 Filtering

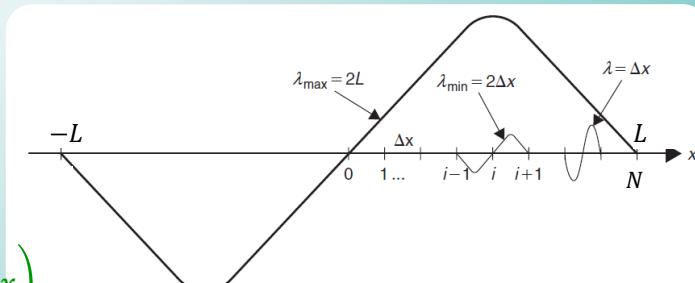
### Implicit filtering – numerical diffusion

- Pure advection test:  $\frac{\partial U}{\partial t} + a \frac{\partial U}{\partial x} = 0$

$$\sigma = \frac{a \Delta t}{\Delta x}$$

$$U_0 = \sin\left(\frac{\phi}{\Delta x} x\right)$$

$$\phi = \frac{j\pi}{N}; j = 0, \dots, N \quad (0 < \phi < \pi)$$



▲ Fourier transform  
of the solution

Chap 6

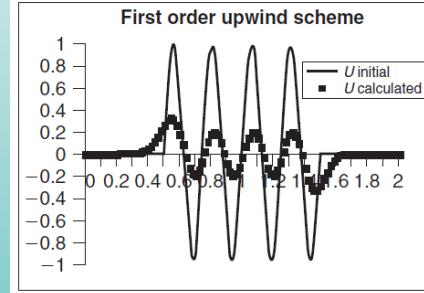
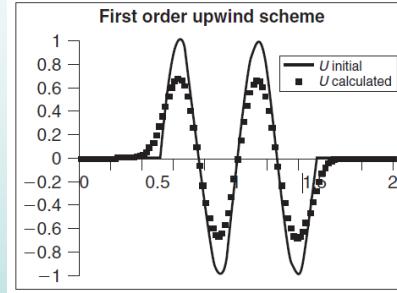
By E. Amani

## VI.1 Filtering

### Implicit filtering – numerical diffusion

- Pure advection test: after 80 time steps

$$\sigma = \frac{a\Delta t}{\Delta x} = 0.8, \quad \phi = \frac{j\pi}{N}$$



Chap 6

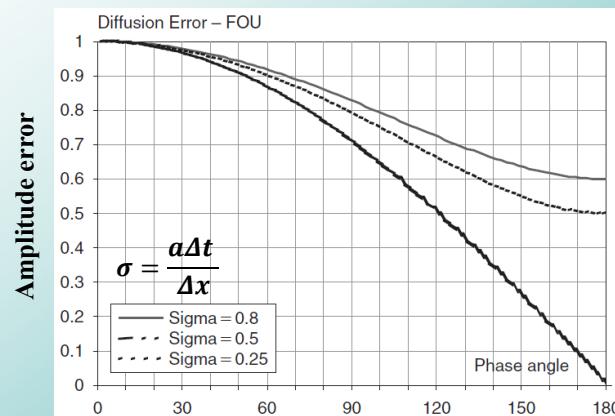
$$\phi = \frac{j\pi}{N} = \frac{8\pi}{100} \blacktriangle \quad \blacktriangle \quad \phi = \frac{j\pi}{N} = \frac{16\pi}{100}$$

By E. Amani

## VI.1 Filtering

### Implicit filtering – numerical diffusion

- Pure advection test:



Chap 6

$$\phi = \frac{j\pi}{N}$$

By E. Amani

## VI.1 Filtering

### Implicit filtering – numerical diffusion

- **Note 1:** For explicit filtering use small  $h/\Delta$
- **Note 2:** In many cases, only implicit filter  $\Delta = h$
- **Note 3:** A non-uniform grid results in a non-homogeneous implicit filter

Chap 6

By E. Amani

## VI.2 Filtered equations

### Taking filter of the Navier-Stokes equations:

$$\frac{\partial \bar{U}_j}{\partial x_j} = 0 \quad (6.28)$$

$$\frac{\partial \bar{U}_i}{\partial t} + \frac{\partial \bar{U}_i \bar{U}_j}{\partial x_j} = \nu \frac{\partial^2 \bar{U}_i}{\partial x_j \partial x_j} - \frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} \quad (6.29)$$

- The term  $\bar{U}_i \bar{U}_j$  raises the closure problem

$$\bar{U}_i \bar{U}_j \equiv \bar{U}_i \bar{U}_j + \tau_{ij}^R \quad \tau_{ij}^R \equiv \bar{U}_i \bar{U}_j - \bar{U}_i \bar{U}_j \quad (6.30)$$

→ Residual or SubGrid-Scale (SGS) stress

$$\frac{\partial \bar{U}_i}{\partial t} = \nu \frac{\partial^2 \bar{U}_i}{\partial x_j \partial x_j} - \frac{\partial \tau_{ij}^R}{\partial x_j} - \frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} \quad (6.31) \quad \frac{\partial}{\partial t} \equiv \frac{\partial}{\partial t} + \vec{U} \cdot \vec{\nabla}$$

- Question:  $\bar{U}_i$  depends on the filter type and width. Where is the effect of these factors in Eqs. (6.28) and (6.31)?

Chap 6

By E. Amani

## VI.2 Filtered equations

**The filtered fluctuation:**

- Remember      Resolved part      Similarly

$$u_i \equiv U_i - \langle U_i \rangle \quad \bar{u}_i \equiv \bar{U}_i - \langle \bar{U}_i \rangle \quad (6.19)$$

$$R_{ij}(\vec{r}) \equiv \langle u_i(\vec{x}) u_j(\vec{x} + \vec{r}) \rangle \quad R_{ij}^R(\vec{r}) \equiv \langle \bar{u}_i(\vec{x}) \bar{u}_j(\vec{x} + \vec{r}) \rangle \quad (6.20)$$

Chap 6

By E. Amani

## VI.2 Filtered equations

**Decomposition of kinetic energies:**

- Turbulent kinetic energy:

$$k = \bar{k} + k' \quad \bar{k} = \frac{1}{2} \langle \bar{u}_i \bar{u}_i \rangle, k' = \frac{1}{2} \langle u_i u_i - \bar{u}_i \bar{u}_i \rangle \quad (6.49)'$$

filtered      subfilter

$$k = k^R + k^{\text{SFS}} \quad k^R = \frac{1}{2} \langle \bar{u}_i \bar{u}_i \rangle, k^{\text{SFS}} = \frac{1}{2} \langle u_i u_i - \bar{u}_i \bar{u}_i \rangle \quad (6.50)'$$

resolved      Unresolved (subfilter-scale)

$$\bar{k} = k^R + k_r \quad k_r = \frac{1}{2} \langle \bar{u}_i \bar{u}_i - \bar{u}_i \bar{u}_i \rangle \quad (6.51)' \quad (k^{\text{SFS}} = k' + k_r)$$

residual

Chap 6

By E. Amani

## VI.2 Filtered equations

### Decomposition of kinetic energies:

- (Instantaneous) kinetic energy:

$$E = \bar{E} + E' \quad \bar{E} = \frac{1}{2} \overline{U_i U_i}, E' = \frac{1}{2} (U_i U_i - \overline{U_i U_i}) \quad (6.49)''$$

$$E = E^R + E^{\text{SFS}} \quad E^R = \frac{1}{2} \overline{U_i \bar{U}_i}, E^{\text{SFS}} = \frac{1}{2} (U_i U_i - \bar{U}_i \bar{U}_i) \quad (6.50)''$$

or  $E_f$

$$\bar{E} = E^R + E_r \quad E_r = \frac{1}{2} (\overline{U_i U_i} - \bar{U}_i \bar{U}_i) \quad (6.51)' \quad (E^{\text{SFS}} = E' + E_r)$$

or  $K_r$  residual

$\underbrace{\frac{1}{2} \tau_{ii}^R}_{\frac{1}{2} \tau_{ii}^R}$

$$\bar{K} \equiv \frac{1}{2} \langle \bar{U}_i \rangle \langle \bar{U}_i \rangle \quad (6.52)''$$

Chap 6

By E. Amani

## VI.2 Filtered equations

### Decomposition of kinetic energies:

- Exercise: show that

$$U_i = \langle \bar{U}_i \rangle + \bar{u}_i + U'_i \quad (6.54)''$$

$$U_i = \langle U_i \rangle + \bar{u}_i + u'_i \quad (6.55)''$$

$$\langle E^R \rangle = \bar{K} + k^R \quad (6.56)''$$

- Transport equation for  $E^R$  [1]:

$$\frac{\bar{D}E^R}{\bar{D}t} - \frac{\partial}{\partial x_j} \left[ \bar{U}_i \left( 2\nu \bar{S}_{ij} - \tau_{ij}^r - \frac{\bar{p} \delta_{ij}}{\rho} \right) \right] = -\mathcal{P}_r - \varepsilon_f \quad (6.57)''$$

$\downarrow \tau_{ii}^R - \frac{1}{3} \tau_{kk}^R \delta_{ij}$

$$\varepsilon_f = 2\nu \bar{S}_{ij} \bar{S}_{ij} \quad (6.58)''$$

$$\mathcal{P}_r = -\tau_{ij}^r \bar{S}_{ij} \quad (6.59)''$$

Chap 6

By E. Amani

## VI.2 Filtered equations

### Decomposition of kinetic energies:

- Transport equation for  $E_r$  [dyn-keq.pdf]:

$$\frac{\bar{D}E_r}{Dt} + \frac{\partial}{\partial x_j} \left[ \frac{1}{2} \bar{U}_i \bar{U}_i \bar{U}_j - \frac{1}{2} \bar{U}_i \bar{U}_i \bar{U}_j + \frac{1}{\rho} \bar{p} \bar{U}_j - \frac{1}{\rho} \bar{p} \bar{U}_j - \bar{U}_i \tau_{ij}^r \right] - \quad (6.60)''$$

$$\nu \frac{\partial^2 E_r}{\partial x_j \partial x_j} = +\mathcal{P}_r - \varepsilon_r$$

$$\varepsilon_r = \nu \left( \frac{\partial \bar{U}_i}{\partial x_j} \frac{\partial \bar{U}_i}{\partial x_j} - \frac{\partial \bar{U}_i}{\partial x_j} \frac{\partial \bar{U}_i}{\partial x_j} \right) \quad (6.91)''$$

Chap 6

By E. Amani

## VI.2 Filtered equations

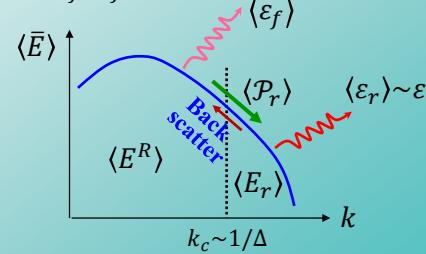
### Decomposition of kinetic energies:

- Energy cascade:

$$\frac{\bar{D}E^R}{Dt} - \frac{\partial}{\partial x_j} \left[ \bar{U}_i \left( 2\nu \bar{S}_{ij} - \tau_{ij}^r - \frac{\bar{p} \delta_{ij}}{\rho} \right) \right] = -\mathcal{P}_r - \varepsilon_f \quad (6.57)''$$

$$\frac{\bar{D}E_r}{Dt} + \frac{\partial}{\partial x_j} \left[ \frac{1}{2} \bar{U}_i \bar{U}_i \bar{U}_j - \frac{1}{2} \bar{U}_i \bar{U}_i \bar{U}_j + \frac{1}{\rho} \bar{p} \bar{U}_j - \frac{1}{\rho} \bar{p} \bar{U}_j - \bar{U}_i \tau_{ij}^r \right] - \quad (6.60)''$$

$$\nu \frac{\partial^2 E_r}{\partial x_j \partial x_j} = +\mathcal{P}_r - \varepsilon_r$$



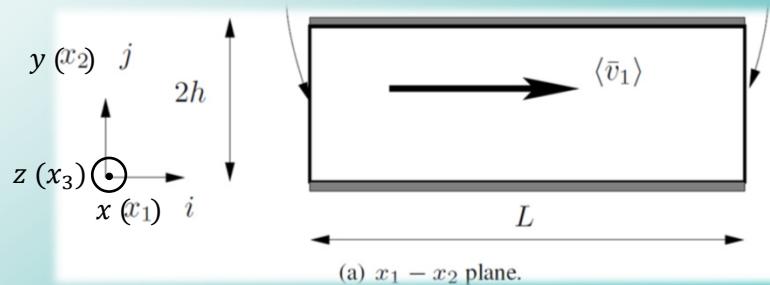
Chap 6

By E. Amani

## Project#1: *A priori* study

### Part I: Filtering the DNS data

- Calculate filtered quantities from DNS data
- Check reference [2] for discretized 3D box filter (**uniform grid!**)



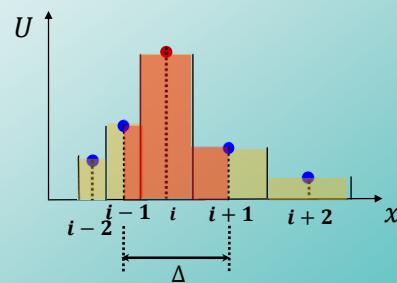
Chap 6

By E. Amani

## Project#1: *A priori* study

### Part I: Filtering the DNS data

- Calculate filtered quantities from DNS data
- Check reference [2] for discretized 3D box filter (**uniform grid!**)
  - Numerical integration: Simple midpoint



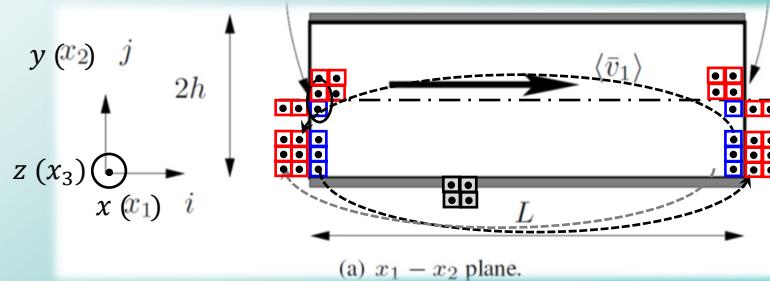
Chap 6

By E. Amani

## Project#1: *A priori* study

### Part I: Filtering the DNS data

- Calculate filtered quantities from DNS data
- Check reference [2] for discretized 3D box filter (**uniform grid!**)
- To keep it simple, do not need to treat the boundary cells
- Only the wall adjacent cells for the filtering operation: Use ghost cells of volume zero!



Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### The Smagorinsky model

- The **eddy-viscosity** assumption
- Remember:

$$\langle u_i u_j \rangle - \frac{2}{3} k \delta_{ij} = -2 \nu_T \langle S_{ij} \rangle \quad (5.11)$$

$\underbrace{\frac{1}{3} \langle u_k u_k \rangle}_{\tau_{ij}^r}$

- Similarly,

$$\tau_{ij}^R - \frac{1}{3} \tau_{kk}^R \delta_{ij} \equiv -2 \nu_r \bar{S}_{ij} \quad (6.40)$$

$\underbrace{\tau_{ij}^r}_{\text{SGS eddy viscosity}}$

$$\bar{S}_{ij} = \frac{1}{2} \left( \frac{\partial \bar{U}_i}{\partial x_j} + \frac{\partial \bar{U}_j}{\partial x_i} \right) \quad (6.41)$$

Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### The Smagorinsky model

- **Exercise:** Substituting Eq. (6.40) into Eq. (6.31), show that

$$\frac{\partial \bar{U}_i}{\partial t} + \frac{\partial \bar{U}_i \bar{U}_j}{\partial x_j} = -\frac{1}{\rho} \frac{\partial \bar{p}}{\partial x_i} + \frac{\partial}{\partial x_j} [2(\nu + \nu_r) \bar{S}_{ij}] \quad (6.42)$$

where  $\bar{p} + \rho \tau_{kk}^R / 3 \rightarrow \bar{p}$

- $\nu_r = ?$
- The kinetic theory of gasses for the molecular viscosity:

$$\nu = \nu \ell$$

Standard deviation of molecular velocity      Mean free path

- Prandtl's analogy for the turbulent viscosity (RANS):

$$\nu_t = \nu_m \ell_m \quad (6.43)$$

**Chap 6**

**By E. Amani**

## VI.3 SubGrid-Scale (SGS) modeling

### The Smagorinsky model

- A simple estimate for  $\nu_m$  and  $\ell_m$  in free shear flows:

➡ **Lecture Notes: 6.3.1**

**Chap 6**

**By E. Amani**

## VI.3 SubGrid-Scale (SGS) modeling

### The Smagorinsky model

- A simple estimate for  $v_m$  and  $\ell_m$  in free shear flows:

$$v_t = \ell_m v_m = \ell_m^2 \left| \frac{d\langle U \rangle}{dy} \right| \quad (6.44)$$

- Smagorinsky's relation for  $\nu_r$ :

$$\nu_r = (C_s \Delta)^2 \bar{S} \quad (6.48) \quad \bar{S} = (2\bar{S}_{ij}\bar{S}_{ij})^{1/2}$$

↗ Smagorinsky constant

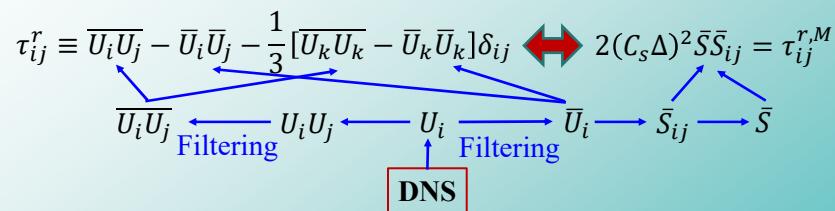
Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

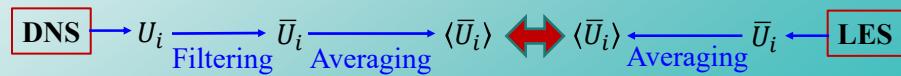
### Tests of model performances

- *A priori* test: Using DNS data



↗ The correlation coefficient is usually used as a measure

- *A posteriori* test: LES



Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### The Smagorinsky model issues

- *A priori* tests show that the Smagorinsky model is too dissipative, e.g.,  $C_s = 0.065 - 0.1$  for channel flows
- The Smagorinsky model is not valid near walls ( $C_s \not\rightarrow 0$ )
- A possible remedy: Van Driest damping function

$$\ell_s = \min(C_s \Delta, \kappa y [1 - \exp(y^+ / A^+)]), \quad A^+ = 26 \quad (6.48)'$$

- Exercise: The derivation of Eq. (6.48)'
- Better solutions?

Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### The dynamic Smagorinsky model

➡ Lecture Notes: 6.3.3

Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Alternative SGS modeling

- Remember:

$$\overline{U_i U_j} \equiv \underbrace{\bar{U}_i \bar{U}_j}_{\text{Resolved part}} + \underbrace{\tau_{ij}^R}_{\text{Unresolved part}} \quad (6.30)$$

Resolved part      Unresolved part      → Modeling

- Instead of directly modeling  $\tau_{ij}^R$ , Germano's decomposition can be used (recommended):

$$\tau_{ij}^R = \mathcal{L}_{ij}^0 + \mathcal{C}_{ij}^0 + \mathcal{R}_{ij}^0 \quad (6.57)$$

Leonard stress      Cross stress      SGS Reynolds stress

$$\mathcal{L}_{ij}^0 = \overline{U_i \bar{U}_j} - \bar{\overline{U}}_i \bar{\overline{U}}_j \quad (6.58) \quad (\text{resolved})$$

$$\mathcal{C}_{ij}^0 = \overline{\bar{U}_i U'_j} + \overline{\bar{U}_j U'_i} - \bar{\overline{U}}_i \bar{U}'_j - \bar{\overline{U}}_j \bar{U}'_i \quad (6.59) \quad (\text{unresolved})$$

$$\mathcal{R}_{ij}^0 = \overline{U'_i U'_j} - \bar{U}'_i \bar{U}'_j \quad (6.60) \quad (\text{unresolved})$$

Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Alternative SGS modeling

- Exercise:** Verify Eq. (6.57).
- All terms of Eq. (6.57) is Galilean invariant like  $\tau_{ij}^R$
- Exercise:** Show that for  $\tilde{\Delta} = \bar{\Delta}$ , the Leonard stress is the same as the resolved SDS stress.
- Using Germano's decomposition:

$$\overline{U_i U_j} \equiv \underbrace{\bar{U}_i \bar{U}_j}_{\text{Resolved}} + \underbrace{\mathcal{L}_{ij}^0 + \mathcal{C}_{ij}^0 + \mathcal{R}_{ij}^0}_{\tau_{ij}^k \text{ (Unresolved)}} \longrightarrow \text{Modeling} \quad (6.61)$$

- Using the dynamic Smagorinsky model for  $\tau_{ij}^k$  results in:

$$\tau_{ij}^r = \left( \mathcal{L}_{ij}^0 - \frac{1}{3} \mathcal{L}_{kk}^0 \delta_{ij} \right) - \underbrace{2 C_{DB} \bar{\Delta}^2 \bar{S} \bar{S}_{ij}}_{\tau_{ij}^k} \quad (6.62)$$

Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Alternative SGS modeling

- The dynamic Bardina model:

$$\tau_{ij}^r = \left( \mathcal{L}_{ij}^0 - \frac{1}{3} \mathcal{L}_{kk}^0 \delta_{ij} \right) - \underbrace{2C_{DB} \bar{\Delta}^2 \bar{S} \bar{S}_{ij}}_{\tau_{ij}^k}, \quad (6.62)$$

- Exercise: It can be shown that:

$$C_{DB} = \frac{(M_{ij} l_{ij}^B)_{ave}}{(M_{kl} M_{kl})_{ave}} \quad (6.56)'$$

$$l_{ij}^B = l_{ij} - H_{ij} \quad (6.63)$$

$$H_{ij} = \widetilde{\overline{U}_i \overline{U}_j} - \underbrace{\widetilde{\overline{U}_i} \widetilde{\overline{U}_j}}_{\mathcal{L}_{ij}^0} - \left( \widetilde{\overline{U}_i \overline{U}_j} - \widetilde{\overline{U}_i} \widetilde{\overline{U}_j} \right) \quad (6.63)'$$

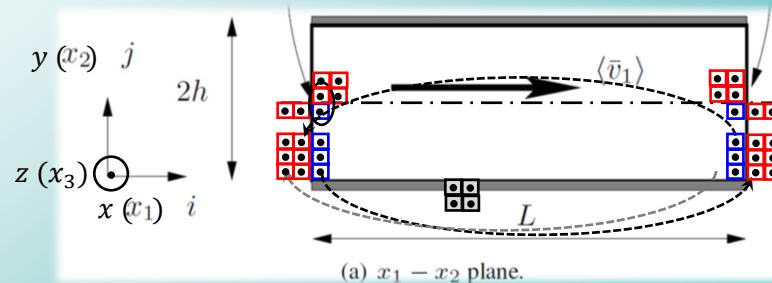
Chap 6

By E. Amani

## Project#1: *A priori* study

### Part II: SGS modeling

- Evaluate the performance static and dynamic SGS models
- Understanding energy cascade in a LES
- Back scatter
- The quality criteria of LES

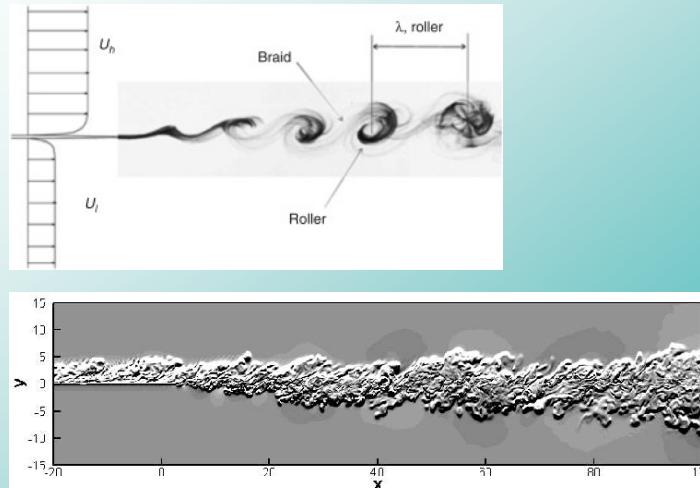


Chap 6

By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Case study: Turbulent mixing layer



Chap 6

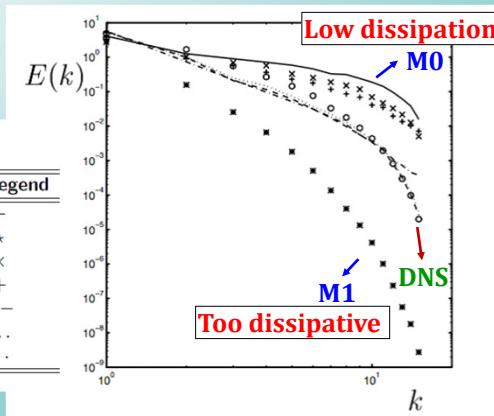
By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Case study: Turbulent mixing layer

An example from Geurts (2004)

Name	Model for $\tau_{ij}$	Plot legend
M0	No model	—
M1	Smagorinsky	★
M2	Similarity	×
M3	Nonlinear	+
M4	Dynamic Smagorinsky	---
M5	Dynamic Mixed	...
M6	Dynamic Nonlinear	—.



Chap 6

▲ (Filtered) DNS (o) and  
LES models (M0-M6). By E. Amani

## VI.3 SubGrid-Scale (SGS) modeling

### Beyond eddy-viscosity models

- Approximate Deconvolution Models (ADM)

The image shows a thumbnail of a journal article from the *Physics of Fluids* journal. The article is titled "Novel mixed approximate deconvolution subgrid-scale models for large-eddy simulation". It is an **ARTICLE** published online on 27 August 2024. The authors listed are Ehsan Amani, Mohammad Bagher Molaei, and Morteza Ghorbani. The thumbnail includes standard journal navigation icons for "View Online", "Export Options", and "Citation".

- ... (Exercise)

Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

- Objective

- Numerical solution of **filtered** equations + B.C. and I.C.

Chap 6

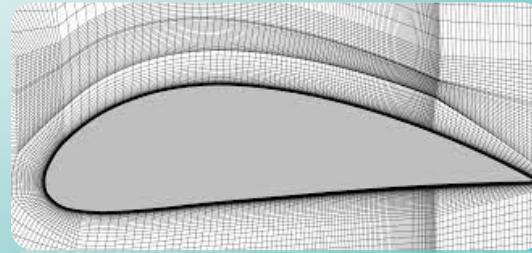
By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

- **Steps**

- 1) Computational grid:
  - Structured (**simple geometries**)



Chap 6

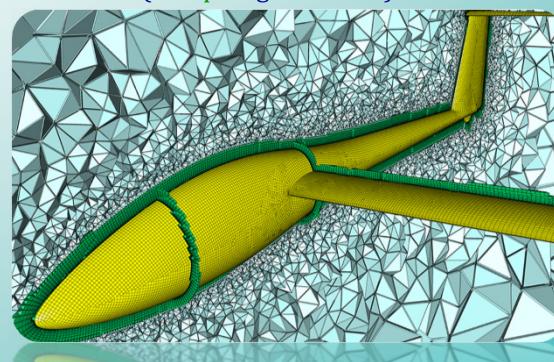
By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

- **Steps**

- 1) Computational grid:
  - Structured (**simple geometries**)
  - Unstructured (**Complex geometries**)



Chap 6

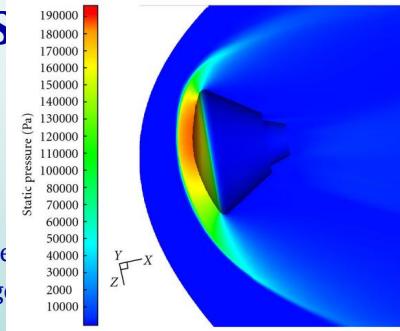
By E. Amani

## VI.4 Numerics of LES

### VI.4.1 Overview of CFD

#### • Steps

- 1) Computational grid:
  - Structured (simple geometries)
  - Unstructured (Complex geometries)
- 2) Solution algorithm:
  - Density-based (High-Mach-number flows)



Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

#### • Steps

- 1) Computational grid:
  - Structured (simple geometries)
  - Unstructured (Complex geometries)
- 2) Solution algorithm:
  - Density-based (High-Mach-number flows)
  - Pressure-based (Low-Mach-number flows)
    - Pressure-velocity decoupling
      - Projection
      - SIMPLE
      - ...

Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

#### • Steps

- 1) Computational grid:
  - Structured (simple geometries)
  - Unstructured (Complex geometries)
- 2) Solution algorithm:
  - Density-based (High-Mach-number flows)
  - Pressure-based (Low-Mach-number flows)
    - Pressure-velocity decoupling
    - Odd-even decoupling
      - Staggered grid
      - Collocated grid

Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

#### • Steps

- 1) Computational grid:
  - Structured (simple geometries)
  - Unstructured (Complex geometries)
- 2) Solution algorithm:
  - Density-based (High-Mach-number flows)
  - Pressure-based (Low-Mach-number flows)
- 3) Discretization method:
  - Finite Difference (FD) (Structured grid)
  - Finite Volume (FV)
  - Finite Element (FE)
  - Spectral methods (SM)
  - Mesh-free methods (MF)

Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.1 Overview of CFD

- **Steps**

- 1) Computational grid:
  - Structured (simple geometries)
  - Unstructured (Complex geometries)
- 2) Solution algorithm:
  - Density-based (High-Mach-number flows)
  - Pressure-based (Low-Mach-number flows)
- 3) Discretization method:
- 4) Treating boundary conditions
- 5) Numerical solution of a set of algebraic equations:
  - Gauss-Seidel
  - Preconditioning
  - Multigrid
  - ...

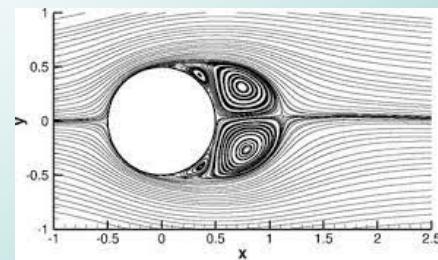
**Chap 6****By E. Amani**

## VI.4 Numerics of LES in physical space

### VI.4.2 CFD considerations – LES vs. RANS

- **Dimensions of the problem**

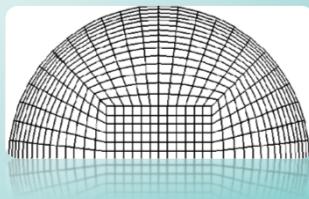
- **LES:** Always 3D time-dependent (realization of fields)
- **RANS:** 2D steady simulation is possible (mean flow fields)

**Chap 6****By E. Amani**

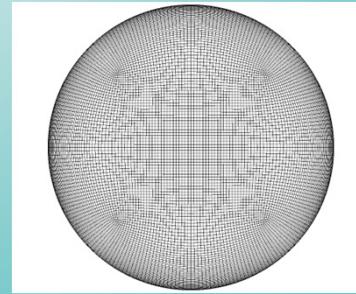
## VI.4 Numerics of LES in physical space

### VI.4.2 CFD considerations – LES vs. RANS

- **Dimensions of the problem**
  - **LES:** Always 3D time-dependent (realization of fields)
  - **RANS:** 2D steady simulation is possible (mean flow fields)
- **Grid**
  - **LES:** Fine but coarser than DNS
  - **RANS:** Coarse



Chap 6



By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.2 CFD considerations – LES vs. RANS

- **Dimensions of the problem**
  - **LES:** Always 3D time-dependent (realization of fields)
  - **RANS:** 2D steady simulation is possible (mean flow fields)
- **Grid**
  - **LES:** Fine but coarser than DNS
  - **RANS:** Coarse
- **Discretization**
  - **LES:** High-order (low-dissipative) central-based (at least 2<sup>nd</sup> order in time and space)
  - **RANS:** Usually upwind-based (stable on coarse grids)
- **Boundary and initial conditions**
  - **LES:** Inflow generators (realization), non-reflective outflows, ...
  - **RANS:** Mean fields are directly set

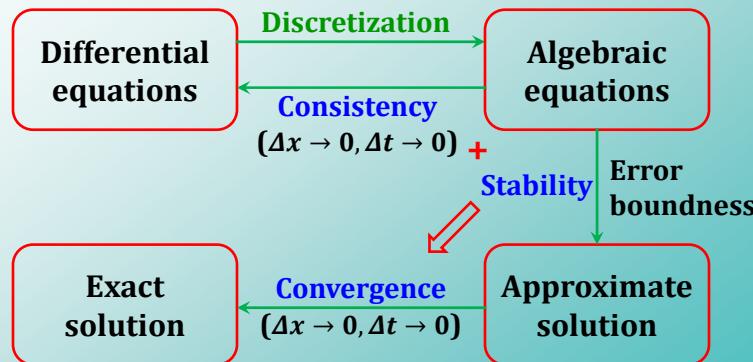
Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.3 The properties of discretization schemes

- Consistency, stability, and convergence



Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.3 The properties of discretization schemes

- Consistency, stability, and convergence
- Accuracy or order of discretization
  - Truncation error in space and time
 
$$\text{Diff. Eq.} = \text{Disc. Eq.} + \text{truncation error } O(\Delta x^n, \Delta t^m)$$
  - FV is limited to  $n = 2$
  - FD and SM (simple geometries) and FE (complex geometries) are preferable for LES

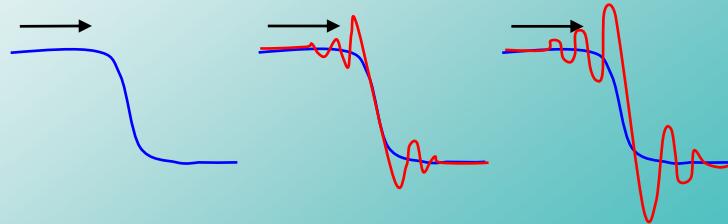
Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.3 The properties of discretization schemes

- **Consistency, stability, and convergence**
- **Accuracy or order of discretization**
- **Monotonicity**
  - Definition: The ability of pure advection of a **monotone high-gradient** solution with no new extrema or unphysical oscillations



Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.3 The properties of discretization schemes

- **Consistency, stability, and convergence**
- **Accuracy or order of discretization**
- **Monotonicity**
  - Definition: The ability of pure advection of a **monotone high-gradient** solution with no new extrema or unphysical oscillations
  - Monotone preserving schemes with non-linear limiters  
**Disc. Eq.** =  $[1 - \Phi(U)] \times \text{FOU} + \Phi(U) \times \text{High-order}$
  - **RANS:** Commonly used
  - **LES:** Great care must be taken (high-quality **grids**, special **low-dissipative** schemes)

Chap 6

By E. Amani

## VI.4 Numerics of LES in physical space

### VI.4.3 The properties of discretization schemes

- **Consistency, stability, and convergence**
- **Accuracy or order of discretization**
- **Monotonicity**
- **Conservativity**
  - Definition: primary and secondary conservative (next section)
  - A favorable property for LES
  - Experience: secondary non-conservative schemes may be **stable** at low **Re** numbers but become **unstable** at higher **Re** numbers

**Chap 6**

**By E. Amani**

## VI.4 Numerics of LES in physical space

### VI.4.4 – VI.4.7

 **Lecture Notes: 6.4.4**

**Chap 6**

**By E. Amani**

## VI.4 Numerics of LES in physical space

### 2<sup>nd</sup>-order fully-conservative scheme on a regular grid

$$(\text{cont.} - R2) \equiv \delta_{2x_j} U_j \quad (6.97)$$

$$(\text{Diff.} - R2)_i \equiv \delta_{1x_j} \tau_{ij} \quad (6.98)$$

$$(\text{Pres.} - R2)_i \equiv \delta_{2x_i} P \quad (6.99)$$

$$(\text{Div.} - R2)_i \equiv \delta_{1x_j} (\bar{U}_j^{1x_j} \bar{U}_i^{1x_j}) \quad (6.101)$$

### 4<sup>nd</sup>-order fully-conservative scheme on a regular grid

$$(\text{cont.} - R4) \equiv \frac{4}{3} \delta_{2x_j} U_j - \frac{1}{3} \delta_{4x_j} U_j \quad (6.102)$$

$$(\text{Diff.} - R4)_i \equiv \frac{4}{3} \delta_{1x_j} \tau_{ij} - \frac{1}{3} \delta_{2x_j} \tau_{ij} \quad (6.103)$$

$$(\text{Pres.} - R4)_i \equiv \frac{4}{3} \delta_{2x_i} P - \frac{1}{3} \delta_{4x_i} P \quad (6.104)$$

$$(\text{Div.} - R4)_i \equiv \frac{4}{3} \delta_{1x_j} (\bar{U}_j^{1x_j} \bar{U}_i^{1x_j}) - \frac{1}{3} \delta_{2x_j} (\bar{U}_j^{2x_j} \bar{U}_i^{2x_j}) \quad (6.105)$$

Chap 6

By E. Amani

## Project#2: *A posteriori* study

### 1D Stochastic Forced Burgers Equation (1DFSBE)

- Evaluate the performance static and dynamic SGS models
- Understanding energy cascade in a LES
- Back scatter
- The quality criteria of LES
- Energy spectrum (Q11 after section VII.1)
- “pyBurgers.py” (ver. 3) code amendment (OOP)

$$\frac{\partial U}{\partial t} + U \frac{\partial U}{\partial x} = \nu \frac{\partial^2 U}{\partial x^2} + \eta(x, t) \quad 0 \leq x \leq 2\pi \quad U(x, 0) = 0$$

$$\eta(x, t) = \sqrt{\frac{2D}{\Delta t}} F^{-1} \left\{ |k|^{\frac{\beta}{2}} \hat{f}(k) \right\}$$

Chap 6

By E. Amani

## Code “pyBurgers.py”, Ver 3

```

File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\LEGION\Desktop\pyBurgers-master\pyBurgers.py
pyBurgers.py X burgerslib.py X namelist.json X
1 #####=====
2 # Code "pyBurgers.py"
3 # 1D Stochastic Burgers' Equation (SBE) LES solver
4 # Version 2, November 2019, by Jeremy Gibbs,
5 # homepage: http://gibbs.science/
6 # Version 3, February 2023, by Ehsan Amani,
7 # homepage: https://sites.google.com/view/dramani
8 # Version 4, date?, by your name?
9 ####===
10 |
11 #!/usr/bin/env python
12 import time
13 from sys import stdout
14 import numpy as np
15 import netCDF4 as nc
16 from burgerslib import Utils, Settings, BurgersLES
17
18 # Main
19 def main():
20
21     # let's time this thing
22     t1 = time.time()
23
24     # a nice welcome message
25     print("#####")
26     print("#")
27     print("#           Welcome to pyBurgers      #")
28     print("#           A fun tool to study turbulence using DNS and LES      #")
29     print("#")
30     print("#####")
31     print("[pyBurgers: Info] \t You are running in LES mode")
32

```

**Chap 6**

**By E. Amani**

## Code “pyBurgers.py”, Ver 3

```

32
33     # instantiate helper classes
34     print("[pyBurgers: Setup] \t Reading input settings")
35     utils = Utils()
36     settings = Settings('namelist.json')
37
38     # input settings
39     nxDNS = settings.nxDNS
40     nxLES = settings.nxLES
41     mp   = int(nxLES/2)
42     dx   = 2*np.pi/nxLES
43     dt   = settings.dt
44     nt   = settings.nt
45     model = settings.sgs
46     visc  = settings.visc
47     damp  = settings.damp
48     beta  = settings.beta
49     nInfo = settings.nInfo
50     nStat = settings.nStat
51     disTy = settings.disTy
52
53     # instantiate the LES SGS class
54     LES = BurgersLES(model)
55
56     # initialize velocity field
57     print("[pyBurgers: Setup] \t Initializing velocity field")
58     u = np.zeros(nxLES)
59
60     # initialize subgrid tke if using Deardorff
61     if model==4:
62         kr = np.ones(nxLES)
63     else:
64         kr = 0
65
66     # initialize random number generator
67     np.random.seed(1)

```

**Chap 6**

**By E. Amani**

## Code “pyBurgers.py”, Ver 3

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\LEGION\Desktop\pyBurgers-master\burgerslib.py

pyBurgers.py X burgerslib.py X namelist.json X

```

1 #=====
2 # Code "pyBurgers.py"
3 # 1D Stochastic Burgers Equation (SBE) DNS/LES library
4 # Version 2, November 2019, by Jeremy Gibbs,
5 # homepage: http://gibbs.science/
6 # Version 3, February 2023, by Ehsan Amani,
7 # homepage: https://sites.google.com/view/dramani
8 # Version 4, date?, by your name?
9 #=====
10
11 import json
12 import numpy as np
13 import cmath as cm
14 import netCDF4 as nc
15 from scipy.stats import norm
16
17 # class with helper utilities
18 class Utils:
19
20     # function to generate fractional Brownian motion (FBM) noise
21     def noise(self, alpha, n):
22         x = np.sqrt(n)*norm.ppf(np.random.rand(n))
23         m = int(n/2)
24         k = np.abs(np.fft.fftfreq(n, d=1/n))
25         k[0] = 1
26         fx = np.fft.fft(x)
27         fx[0] = 0
28         fx[m] = 0
29         fx1 = fx * ( k**(-alpha/2) )
30         x1 = np.real(np.fft.ifft(fx1))
31
32         return x1
33
34

```

**Forcing function**

**Chap 6**

**By E. Amani**

## Code “pyBurgers.py”, Ver 3

File Edit Search Source Run Debug Consoles Projects Tools View Help

C:\Users\LEGION\Desktop\pyBurgers-master\burgerslib.py

pyBurgers.py X burgerslib.py X namelist.json X

```

159
160     # class to read input settings
161     class Settings:
162
163         # initializer to get settings from provided namelist
164         # visc: Kinematic Viscosity; damp: Diffusivity of Noise; beta: Spectrum slope
165         # nxLES: Number of grid points; dx: Grid Spacing; nt: Total number of Iterations
166         # dt: Time step; nxDNS: Number of DNS grid points
167         # nInfo: Info output frequency on screen; nStat: Statistics output frequency
168         # numerics dtype: 0 (spectral); 1 (physical)
169         def __init__(self, namelist):
170             with open(namelist) as json_file:
171                 data = json.load(json_file)
172                 self.nxLES = data["les"]["nx"]
173                 self.sgs = data["les"]["sgs"]
174                 self.disTy = data["num"]["disTy"]
175                 self.nt = data["nt"]
176                 self.dt = data["dt"]
177                 self.visc = data["visc"]
178                 self.damp = data["damp"]
179                 self.beta = data["beta"]
180                 self.nInfo = data["nInfo"]
181                 self.nStat = data["nStat"]
182
183

```

0: Spectral

2: 2<sup>nd</sup>-order fully-conservative

4: 4<sup>th</sup>-order fully-conservative

**Chap 6**

**By E. Amani**

## Code “pyBurgers.py”, Ver 3

No SGS  
stress

```

183
184 # class to model subgrid terms
185 class BurgersLES:
186
187     # initializer to get selected subgrid model
188     def __init__(self,model):
189         self.model = model
190         settings = Settings('namelist.json')
191         self.dt = settings.dt
192         self.distY = settings.distY
193         if self.model==0:
194             print("[pyBurgers: SGS] \t Running with no model")
195         if self.model==1:
196             print("[pyBurgers: SGS] \t Constant-coefficient Smagorinsky")
197         if self.model==2:
198             print("[pyBurgers: SGS] \t Dynamic Smagorinsky")
199         if self.model==3:
200             print("[pyBurgers: SGS] \t Dynamic Wong-Lilly")
201         if self.model==4:
202             print("[pyBurgers: SGS] \t Deardorff 1.5-order TKE")
203
204     # function to compute subgrid terms in Fourier space
205     def subgrid(self,u,dudx,dx,kn):
206
207         # signal size information
208         n = int(u.shape[0])

```

Chap 6

By E. Amani

## Code “pyBurgers.py”, Ver 3

→ Accessing  
object  
data

→ Accessing  
object  
functions

Chap 6

By E. Amani

```

32
33     # instantiate helper classes
34     print("[pyBurgers: Setup] \t Reading input settings")
35     utils = Utils()
36     settings = Settings('namelist.json')
37
38     # input settings
39     nxDNS = settings.nxDNS
40     nxLES = settings.nxLES
41     mp   = int(nxLES/2)
42     dx   = 2*np.pi/nxLES
43     dt   = settings.dt
44     nt   = settings.nt
45     model = settings.sgs
46     visc = settings.visc
47     damp = settings.damp
48     beta = settings.beta
49     nInfo = settings.nInfo
50     nStat = settings.nStat
51     distY = settings.distY
52
53     # instantiate the LES SGS class
54     LES = BurgersLES(model)
55
56     # initialize velocity field
57     print("[pyBurgers: Setup] \t Initializing velocity field")
58     u = np.zeros(nxLES)
59
60     # initialize subgrid tke if using Deardorff
61     if model==4:
62         kr = np.ones(nxLES)
63     else:
64         kr = 0
65
66     # initialize random number generator
67     np.random.seed(1)

```

## Code “pyBurgers.py”, Ver 3

### Initialization, output files, and grid

```

55      # initialize velocity field
56      print("[pyBurgers: Setup] \t Initializing velocity field")
57      u = np.zeros(nxLES)
58
59      # initialize subgrid tke if using Deardorff
60      if model==4:
61          kr = np.ones(nxLES)
62      else:
63          kr = 0
64
65      # initialize random number generator
66      np.random.seed(1)
67
68      # place holder for right hand side
69      rhsp = 0
70      diff = 0
71      conv = 0
72      fbfm = 0
73      RSG = 0
74
75      # create output file
76      print("[pyBurgers: Setup] \t Creating output file")
77      output = nc.Dataset('pyBurgersLES.nc','w')
78
79      # write x data
80      out_x[:] = np.arange(0,2*np.pi,dx)
81
82      # write time data
83      out_t[:] = np.arange(0,tEnd,dt)
84
85      # write k data
86      out_k[:] = np.arange(0,kEnd,dk)
87
88      # write kr data
89      out_kr[:] = np.arange(0,krEnd,dkr)
90
91      # write u data
92      out_u[:] = u
93
94      # write rhsp data
95      out_rhsp[:] = rhsp
96
97      # write diff data
98      out_diff[:] = diff
99
100     # write conv data
101     out_conv[:] = conv
102
103     # write fbfm data
104     out_fbfm[:] = fbfm
105
106     # write RSG data
107     out_RSG[:] = RSG
108
109     # write noise data
110     out_noise[:] = noise
111
112     # write viscosity data
113     out_viscosity[:] = viscosity
114
115     # write dampening data
116     out_dampening[:] = dampening
117
118     # write beta data
119     out_beta[:] = beta
120
121     # write dx data
122     out_dx[:] = dx
123
124     # write dt data
125     out_dt[:] = dt
126
127     # write dk data
128     out_dk[:] = dk
129
130     # write dkr data
131     out_dkr[:] = dkr
132
133     # write nxLES data
134     out_nxLES[:] = nxLES
135
136     # write nxDNS data
137     out_nxDNS[:] = nxDNS
138
139     # write disT data
140     out_disT[:] = disT
141
142     # write derivs data
143     out_derivs[:] = derivs
144
145     # write dudx data
146     out_dudx[:] = dudx
147
148     # write du2dx data
149     out_du2dx[:] = du2dx
150
151     # write d2udx2 data
152     out_d2udx2[:] = d2udx2
153
154     # write d3udx3 data
155     out_d3udx3[:] = d3udx3
156
157     # write conv data
158     out_conv[:] = conv
159
160     # write diff data
161     out_diff[:] = diff
162
163     # write fbfm data
164     out_fbfm[:] = fbfm
165
166     # write RSG data
167     out_RSG[:] = RSG
168
169     # write noise data
170     out_noise[:] = noise
171
172     # write viscosity data
173     out_viscosity[:] = viscosity
174
175     # write dampening data
176     out_dampening[:] = dampening
177
178     # write beta data
179     out_beta[:] = beta
180
181     # write dx data
182     out_dx[:] = dx
183
184     # write dt data
185     out_dt[:] = dt
186
187     # write dk data
188     out_dk[:] = dk
189
190     # write dkr data
191     out_dkr[:] = dkr
192
193     # write nxLES data
194     out_nxLES[:] = nxLES
195
196     # write nxDNS data
197     out_nxDNS[:] = nxDNS
198
199     # write disT data
200     out_disT[:] = disT
201
202     # write derivs data
203     out_derivs[:] = derivs
204
205     # write dudx data
206     out_dudx[:] = dudx
207
208     # write du2dx data
209     out_du2dx[:] = du2dx
210
211     # write d2udx2 data
212     out_d2udx2[:] = d2udx2
213
214     # write d3udx3 data
215     out_d3udx3[:] = d3udx3
216
217     # write conv data
218     out_conv[:] = conv
219
220     # write diff data
221     out_diff[:] = diff
222
223     # write fbfm data
224     out_fbfm[:] = fbfm
225
226     # write RSG data
227     out_RSG[:] = RSG
228
229     # write noise data
230     out_noise[:] = noise
231
232     # write viscosity data
233     out_viscosity[:] = viscosity
234
235     # write dampening data
236     out_dampening[:] = dampening
237
238     # write beta data
239     out_beta[:] = beta
240
241     # write dx data
242     out_dx[:] = dx
243
244     # write dt data
245     out_dt[:] = dt
246
247     # write dk data
248     out_dk[:] = dk
249
250     # write dkr data
251     out_dkr[:] = dkr
252
253     # write nxLES data
254     out_nxLES[:] = nxLES
255
256     # write nxDNS data
257     out_nxDNS[:] = nxDNS
258
259     # write disT data
260     out_disT[:] = disT
261
262     # write derivs data
263     out_derivs[:] = derivs
264
265     # write dudx data
266     out_dudx[:] = dudx
267
268     # write du2dx data
269     out_du2dx[:] = du2dx
270
271     # write d2udx2 data
272     out_d2udx2[:] = d2udx2
273
274     # write d3udx3 data
275     out_d3udx3[:] = d3udx3
276
277     # write conv data
278     out_conv[:] = conv
279
280     # write diff data
281     out_diff[:] = diff
282
283     # write fbfm data
284     out_fbfm[:] = fbfm
285
286     # write RSG data
287     out_RSG[:] = RSG
288
289     # write noise data
290     out_noise[:] = noise
291
292     # write viscosity data
293     out_viscosity[:] = viscosity
294
295     # write dampening data
296     out_dampening[:] = dampening
297
298     # write beta data
299     out_beta[:] = beta
300
301     # write dx data
302     out_dx[:] = dx
303
304     # write dt data
305     out_dt[:] = dt
306
307     # write dk data
308     out_dk[:] = dk
309
310     # write dkr data
311     out_dkr[:] = dkr
312
313     # write nxLES data
314     out_nxLES[:] = nxLES
315
316     # write nxDNS data
317     out_nxDNS[:] = nxDNS
318
319     # write disT data
320     out_disT[:] = disT
321
322     # write derivs data
323     out_derivs[:] = derivs
324
325     # write dudx data
326     out_dudx[:] = dudx
327
328     # write du2dx data
329     out_du2dx[:] = du2dx
330
331     # write d2udx2 data
332     out_d2udx2[:] = d2udx2
333
334     # write d3udx3 data
335     out_d3udx3[:] = d3udx3
336
337     # write conv data
338     out_conv[:] = conv
339
339     # write diff data
340     out_diff[:] = diff
341
342     # write fbfm data
343     out_fbfm[:] = fbfm
344
345     # write RSG data
346     out_RSG[:] = RSG
347
348     # write noise data
349     out_noise[:] = noise
350
350     # write viscosity data
351     out_viscosity[:] = viscosity
352
351     # write dampening data
352     out_dampening[:] = dampening
353
352     # write beta data
353     out_beta[:] = beta
354
353     # write dx data
354     out_dx[:] = dx
355
354     # write dt data
355     out_dt[:] = dt
356
355     # write dk data
356     out_dk[:] = dk
357
356     # write dkr data
357     out_dkr[:] = dkr
358
357     # write nxLES data
358     out_nxLES[:] = nxLES
359
358     # write nxDNS data
359     out_nxDNS[:] = nxDNS
360
359     # write disT data
360     out_disT[:] = disT
361
360     # write derivs data
361     out_derivs[:] = derivs
362
361     # write dudx data
362     out_dudx[:] = dudx
363
362     # write du2dx data
363     out_du2dx[:] = du2dx
364
363     # write d2udx2 data
364     out_d2udx2[:] = d2udx2
365
364     # write d3udx3 data
365     out_d3udx3[:] = d3udx3
366
365     # write conv data
366     out_conv[:] = conv
367
366     # write diff data
367     out_diff[:] = diff
368
367     # write fbfm data
368     out_fbfm[:] = fbfm
369
368     # write RSG data
369     out_RSG[:] = RSG
370
369     # write noise data
370     out_noise[:] = noise
371
370     # write viscosity data
371     out_viscosity[:] = viscosity
372
371     # write dampening data
372     out_dampening[:] = dampening
373
372     # write beta data
373     out_beta[:] = beta
374
373     # write dx data
374     out_dx[:] = dx
375
374     # write dt data
375     out_dt[:] = dt
376
375     # write dk data
376     out_dk[:] = dk
377
376     # write dkr data
377     out_dkr[:] = dkr
378
377     # write nxLES data
378     out_nxLES[:] = nxLES
379
378     # write nxDNS data
379     out_nxDNS[:] = nxDNS
380
379     # write disT data
380     out_disT[:] = disT
381
380     # write derivs data
381     out_derivs[:] = derivs
382
381     # write dudx data
382     out_dudx[:] = dudx
383
382     # write du2dx data
383     out_du2dx[:] = du2dx
384
383     # write d2udx2 data
384     out_d2udx2[:] = d2udx2
385
384     # write d3udx3 data
385     out_d3udx3[:] = d3udx3
386
385     # write conv data
386     out_conv[:] = conv
387
386     # write diff data
387     out_diff[:] = diff
388
387     # write fbfm data
388     out_fbfm[:] = fbfm
389
388     # write RSG data
389     out_RSG[:] = RSG
390
389     # write noise data
390     out_noise[:] = noise
391
390     # write viscosity data
391     out_viscosity[:] = viscosity
392
391     # write dampening data
392     out_dampening[:] = dampening
393
392     # write beta data
393     out_beta[:] = beta
394
393     # write dx data
394     out_dx[:] = dx
395
394     # write dt data
395     out_dt[:] = dt
396
395     # write dk data
396     out_dk[:] = dk
397
396     # write dkr data
397     out_dkr[:] = dkr
398
397     # write nxLES data
398     out_nxLES[:] = nxLES
399
398     # write nxDNS data
399     out_nxDNS[:] = nxDNS
400
400     # write disT data
401     out_disT[:] = disT
402
401     # write derivs data
402     out_derivs[:] = derivs
403
402     # write dudx data
403     out_dudx[:] = dudx
404
403     # write du2dx data
404     out_du2dx[:] = du2dx
405
404     # write d2udx2 data
405     out_d2udx2[:] = d2udx2
406
405     # write d3udx3 data
406     out_d3udx3[:] = d3udx3
407
406     # write conv data
407     out_conv[:] = conv
408
407     # write diff data
408     out_diff[:] = diff
409
408     # write fbfm data
409     out_fbfm[:] = fbfm
410
409     # write RSG data
410     out_RSG[:] = RSG
411
410     # write noise data
411     out_noise[:] = noise
412
411     # write viscosity data
412     out_viscosity[:] = viscosity
413
412     # write dampening data
413     out_dampening[:] = dampening
414
413     # write beta data
414     out_beta[:] = beta
415
414     # write dx data
415     out_dx[:] = dx
416
415     # write dt data
416     out_dt[:] = dt
417
416     # write dk data
417     out_dk[:] = dk
418
417     # write dkr data
418     out_dkr[:] = dkr
419
418     # write nxLES data
419     out_nxLES[:] = nxLES
420
419     # write nxDNS data
420     out_nxDNS[:] = nxDNS
421
420     # write disT data
421     out_disT[:] = disT
422
421     # write derivs data
422     out_derivs[:] = derivs
423
422     # write dudx data
423     out_dudx[:] = dudx
424
423     # write du2dx data
424     out_du2dx[:] = du2dx
425
424     # write d2udx2 data
425     out_d2udx2[:] = d2udx2
426
425     # write d3udx3 data
426     out_d3udx3[:] = d3udx3
427
426     # write conv data
427     out_conv[:] = conv
428
427     # write diff data
428     out_diff[:] = diff
429
428     # write fbfm data
429     out_fbfm[:] = fbfm
430
429     # write RSG data
430     out_RSG[:] = RSG
431
430     # write noise data
431     out_noise[:] = noise
432
431     # write viscosity data
432     out_viscosity[:] = viscosity
433
432     # write dampening data
433     out_dampening[:] = dampening
434
433     # write beta data
434     out_beta[:] = beta
435
434     # write dx data
435     out_dx[:] = dx
436
435     # write dt data
436     out_dt[:] = dt
437
436     # write dk data
437     out_dk[:] = dk
438
437     # write dkr data
438     out_dkr[:] = dkr
439
438     # write nxLES data
439     out_nxLES[:] = nxLES
440
439     # write nxDNS data
440     out_nxDNS[:] = nxDNS
441
440     # write disT data
441     out_disT[:] = disT
442
441     # write derivs data
442     out_derivs[:] = derivs
443
442     # write dudx data
443     out_dudx[:] = dudx
444
443     # write du2dx data
444     out_du2dx[:] = du2dx
445
444     # write d2udx2 data
445     out_d2udx2[:] = d2udx2
446
445     # write d3udx3 data
446     out_d3udx3[:] = d3udx3
447
446     # write conv data
447     out_conv[:] = conv
448
447     # write diff data
448     out_diff[:] = diff
449
448     # write fbfm data
449     out_fbfm[:] = fbfm
450
449     # write RSG data
450     out_RSG[:] = RSG
451
450     # write noise data
451     out_noise[:] = noise
452
451     # write viscosity data
452     out_viscosity[:] = viscosity
453
452     # write dampening data
453     out_dampening[:] = dampening
454
453     # write beta data
454     out_beta[:] = beta
455
454     # write dx data
455     out_dx[:] = dx
456
455     # write dt data
456     out_dt[:] = dt
457
456     # write dk data
457     out_dk[:] = dk
458
457     # write dkr data
458     out_dkr[:] = dkr
459
458     # write nxLES data
459     out_nxLES[:] = nxLES
460
459     # write nxDNS data
460     out_nxDNS[:] = nxDNS
461
460     # write disT data
461     out_disT[:] = disT
462
461     # write derivs data
462     out_derivs[:] = derivs
463
462     # write dudx data
463     out_dudx[:] = dudx
464
463     # write du2dx data
464     out_du2dx[:] = du2dx
465
464     # write d2udx2 data
465     out_d2udx2[:] = d2udx2
466
465     # write d3udx3 data
466     out_d3udx3[:] = d3udx3
467
466     # write conv data
467     out_conv[:] = conv
468
467     # write diff data
468     out_diff[:] = diff
469
468     # write fbfm data
469     out_fbfm[:] = fbfm
470
469     # write RSG data
470     out_RSG[:] = RSG
471
470     # write noise data
471     out_noise[:] = noise
472
471     # write viscosity data
472     out_viscosity[:] = viscosity
473
472     # write dampening data
473     out_dampening[:] = dampening
474
473     # write beta data
474     out_beta[:] = beta
475
474     # write dx data
475     out_dx[:] = dx
476
475     # write dt data
476     out_dt[:] = dt
477
476     # write dk data
477     out_dk[:] = dk
478
477     # write dkr data
478     out_dkr[:] = dkr
479
478     # write nxLES data
479     out_nxLES[:] = nxLES
480
479     # write nxDNS data
480     out_nxDNS[:] = nxDNS
481
480     # write disT data
481     out_disT[:] = disT
482
481     # write derivs data
482     out_derivs[:] = derivs
483
482     # write dudx data
483     out_dudx[:] = dudx
484
483     # write du2dx data
484     out_du2dx[:] = du2dx
485
484     # write d2udx2 data
485     out_d2udx2[:] = d2udx2
486
485     # write d3udx3 data
486     out_d3udx3[:] = d3udx3
487
486     # write conv data
487     out_conv[:] = conv
488
487     # write diff data
488     out_diff[:] = diff
489
488     # write fbfm data
489     out_fbfm[:] = fbfm
490
489     # write RSG data
490     out_RSG[:] = RSG
491
490     # write noise data
491     out_noise[:] = noise
492
491     # write viscosity data
492     out_viscosity[:] = viscosity
493
492     # write dampening data
493     out_dampening[:] = dampening
494
493     # write beta data
494     out_beta[:] = beta
495
494     # write dx data
495     out_dx[:] = dx
496
495     # write dt data
496     out_dt[:] = dt
497
496     # write dk data
497     out_dk[:] = dk
498
497     # write dkr data
498     out_dkr[:] = dkr
499
498     # write nxLES data
499     out_nxLES[:] = nxLES
500
499     # write nxDNS data
500     out_nxDNS[:] = nxDNS
501
500     # write disT data
501     out_disT[:] = disT
502
501     # write derivs data
502     out_derivs[:] = derivs
503
502     # write dudx data
503     out_dudx[:] = dudx
504
503     # write du2dx data
504     out_du2dx[:] = du2dx
505
504     # write d2udx2 data
505     out_d2udx2[:] = d2udx2
506
505     # write d3udx3 data
506     out_d3udx3[:] = d3udx3
507
506     # write conv data
507     out_conv[:] = conv
508
507     # write diff data
508     out_diff[:] = diff
509
508     # write fbfm data
509     out_fbfm[:] = fbfm
510
509     # write RSG data
510     out_RSG[:] = RSG
511
510     # write noise data
511     out_noise[:] = noise
512
511     # write viscosity data
512     out_viscosity[:] = viscosity
513
512     # write dampening data
513     out_dampening[:] = dampening
514
513     # write beta data
514     out_beta[:] = beta
515
514     # write dx data
515     out_dx[:] = dx
516
515     # write dt data
516     out_dt[:] = dt
517
516     # write dk data
517     out_dk[:] = dk
518
517     # write dkr data
518     out_dkr[:] = dkr
519
518     # write nxLES data
519     out_nxLES[:] = nxLES
520
519     # write nxDNS data
520     out_nxDNS[:] = nxDNS
521
520     # write disT data
521     out_disT[:] = disT
522
521     # write derivs data
522     out_derivs[:] = derivs
523
522     # write dudx data
523     out_dudx[:] = dudx
524
523     # write du2dx data
524     out_du2dx[:] = du2dx
525
524     # write d2udx2 data
525     out_d2udx2[:] = d2udx2
526
525     # write d3udx3 data
526     out_d3udx3[:] = d3udx3
527
526     # write conv data
527     out_conv[:] = conv
528
527     # write diff data
528     out_diff[:] = diff
529
528     # write fbfm data
529     out_fbfm[:] = fbfm
530
529     # write RSG data
530     out_RSG[:] = RSG
531
530     # write noise data
531     out_noise[:] = noise
532
531     # write viscosity data
532     out_viscosity[:] = viscosity
533
532     # write dampening data
533     out_dampening[:] = dampening
534
533     # write beta data
534     out_beta[:] = beta
535
534     # write dx data
535     out_dx[:] = dx
536
535     # write dt data
536     out_dt[:] = dt
537
536     # write dk data
537     out_dk[:] = dk
538
537     # write dkr data
538     out_dkr[:] = dkr
539
538     # write nxLES data
539     out_nxLES[:] = nxLES
540
539     # write nxDNS data
540     out_nxDNS[:] = nxDNS
541
540     # write disT data
541     out_disT[:] = disT
542
541     # write derivs data
542     out_derivs[:] = derivs
543
542     # write dudx data
543     out_dudx[:] = dudx
544
543     # write du2dx data
544     out_du2dx[:] = du2dx
545
544     # write d2udx2 data
545     out_d2udx2[:] = d2udx2
546
545     # write d3udx3 data
546     out_d3udx3[:] = d3udx3
547
546     # write conv data
547     out_conv[:] = conv
548
547     # write diff data
548     out_diff[:] = diff
549
548     # write fbfm data
549     out_fbfm[:] = fbfm
550
549     # write RSG data
550     out_RSG[:] = RSG
551
550     # write noise data
551     out_noise[:] = noise
552
551     # write viscosity data
552     out_viscosity[:] = viscosity
553
552     # write dampening data
553     out_dampening[:] = dampening
554
553     # write beta data
554     out_beta[:] = beta
555
554     # write dx data
555     out_dx[:] = dx
556
555     # write dt data
556     out_dt[:] = dt
557
556     # write dk data
557     out_dk[:] = dk
558
557     # write dkr data
558     out_dkr[:] = dkr
559
558     # write nxLES data
559     out_nxLES[:] = nxLES
560
559     # write nxDNS data
560     out_nxDNS[:] = nxDNS
561
560     # write disT data
561     out_disT[:] = disT
562
561     # write derivs data
562     out_derivs[:] = derivs
563
562     # write dudx data
563     out_dudx[:] = dudx
564
563     # write du2dx data
564     out_du2dx[:] = du2dx
565
564     # write d2udx2 data
565     out_d2udx2[:] = d2udx2
566
565     # write d3udx3 data
566     out_d3udx3[:] = d3udx3
567
566     # write conv data
567     out_conv[:] = conv
568
567     # write diff data
568     out_diff[:] = diff
569
568     # write fbfm data
569     out_fbfm[:] = fbfm
570
569     # write RSG data
570     out_RSG[:] = RSG
571
570     # write noise data
571     out_noise[:] = noise
572
571     # write viscosity data
572     out_viscosity[:] = viscosity
573
572     # write dampening data
573     out_dampening[:] = dampening
574
573     # write beta data
574     out_beta[:] = beta
575
574     # write dx data
575     out_dx[:] = dx
576
575     # write dt data
576     out_dt[:] = dt
577
576     # write dk data
577     out_dk[:] = dk
578
577     # write dkr data
578     out_dkr[:] = dkr
579
578     # write nxLES data
579     out_nxLES[:] = nxLES
580
579     # write nxDNS data
580     out_nxDNS[:] = nxDNS
581
580     # write disT data
581     out_disT[:] = disT
582
581     # write derivs data
582     out_derivs[:] = derivs
583
582     # write dudx data
583     out_dudx[:] = dudx
584
583     # write du2dx data
584     out_du2dx[:] = du2dx
585
584     # write d2udx2 data
585     out_d2udx2[:] = d2udx2
586
585     # write d3udx3 data
586     out_d3udx3[:] = d3udx3
587
586     # write conv data
587     out_conv[:] = conv
588
587     # write diff data
588     out_diff[:] = diff
589
588     # write fbfm data
589     out_fbfm[:] = fbfm
590
589     # write RSG data
590     out_RSG[:] = RSG
591
590     # write noise data
591     out_noise[:] = noise
592
591     # write viscosity data
592     out_viscosity[:] = viscosity
593
592     # write dampening data
593     out_dampening[:] = dampening
594
593     # write beta data
594     out_beta[:] = beta
595
594     # write dx data
595     out_dx[:] = dx
596
595     # write dt data
596     out_dt[:] = dt
597
596     # write dk data
597     out_dk[:] = dk
598
597     # write dkr data
598     out_dkr[:] = dkr
599
598     # write nxLES data
599     out_nxLES[:] = nxLES
600
599     # write nxDNS data
600     out_nxDNS[:] = nxDNS
601
600     # write disT data
601     out_disT[:] = disT
602
601     # write derivs data
602     out_derivs[:] = derivs
603
602     # write dudx data
603     out_dudx[:] = dudx
604
603     # write du2dx data
604     out_du2dx[:] = du2dx
605
604     # write d2udx2 data
605     out_d2udx2[:] = d2udx2
606
605     # write d3udx3 data
606     out_d3udx3[:] = d3udx3
607
606     # write conv data
607     out_conv[:] = conv
608
607     # write diff data
608     out_diff[:] = diff
609
608     # write fbfm data
609     out_fbfm[:] = fbfm
610
609     # write RSG data
610     out_RSG[:] = RSG
611
610     # write noise data
611     out_noise[:] = noise
612
611     # write viscosity data
612     out_viscosity[:] = viscosity
613
612     # write dampening data
613     out_dampening[:] = dampening
614
613     # write beta data
614     out_beta[:] = beta
615
614     # write dx data
615     out_dx[:] = dx
616
615     # write dt data
616     out_dt[:] = dt
617
616     # write dk data
617     out_dk[:] = dk
618
617     # write dkr data
618     out_dkr[:] = dkr
619
618     # write nxLES data
619     out_nxLES[:] = nxLES
620
619     # write nxDNS data
620     out_nxDNS[:] = nxDNS
621
620     # write disT data
621     out_disT[:] = disT
622
621     # write derivs data
622     out_derivs[:] = derivs
623
622     # write dudx data
623     out_dudx[:] = dudx
624
623     # write du2dx data
624     out_du2dx[:] = du2dx
625
624     # write d2udx2 data
625     out_d2udx2[:] = d2udx2
626
625     # write d3udx3 data
626     out_d3udx3[:] = d3udx3
627
626     # write conv data
627     out_conv[:] = conv
628
627     # write diff data
628     out_diff[:] = diff
629
628     # write fbfm data
629     out_fbfm[:] = fbfm
630
629     # write RSG data
630     out_RSG[:] = RSG
631
630     # write noise data
631     out_noise[:] = noise
632
631     # write viscosity data
632     out_viscosity[:] = viscosity
633
632     # write dampening data
633     out_dampening[:] = dampening
634
633     # write beta data
634     out_beta[:] = beta
635
634     # write dx data
635     out_dx[:] = dx
636
635     # write dt data
636     out_dt[:] = dt
637
636     # write dk data
637     out_dk[:] = dk
638
637     # write dkr data
638     out_dkr[:] = dkr
639
638     # write nxLES data
639     out_nxLES[:] = nxLES
640
639     # write nxDNS data
640     out_nxDNS[:] = nxDNS
641
640     # write disT data
641     out_disT[:] = disT
642
641     # write derivs data
642     out_derivs[:] = derivs
643
642     # write dudx data
643     out_dudx[:] = dudx
644
643     # write du2dx data
644     out_du2dx[:] = du2dx
645
644     # write d2udx2 data
645     out_d2udx2[:] = d2udx2
646
645     # write d3udx3 data
646     out_d3udx3[:] = d3udx3
647
646     # write conv data
647     out_conv[:] = conv
648
647     # write diff data
648     out_diff[:] = diff
649
648     # write fbfm data
649     out_fbfm[:] = fbfm
650
649     # write RSG data
650     out_RSG[:] = RSG
651
650     # write noise data
651     out_noise[:] = noise
652
651     # write viscosity data
652     out_viscosity[:] = viscosity
653
652     # write dampening data
653     out_dampening[:] = dampening
654
653     # write beta data
654     out_beta[:] = beta
655
654     # write dx data
655     out_dx[:] = dx
656
655     # write dt data
656     out_dt[:] = dt
657
656     # write dk data
657     out_dk[:] = dk
658
657     # write dkr data
658     out_dkr[:] = dkr
659
658     # write nxLES data
659     out_nxLES[:] = nxLES
660
659     # write nxDNS data
660     out_nxDNS[:] = nxDNS
661
660     # write disT data
661     out_disT[:] = disT
662
661     # write derivs data
662     out_derivs[:] = derivs
663
662     # write dudx data
663     out_dudx[:] = dudx
664
663     # write du2dx data
664     out_du2dx[:] = du2dx
665
664     # write d2udx2 data
665     out_d2udx2[:] = d2udx2
666
665     # write d3udx3 data
666     out_d3udx3[:] = d3udx3
667
666     # write conv data
667     out_conv[:] = conv
668
667     # write diff data
668     out_diff[:] = diff
669
668     # write fbfm data
669     out_fbfm[:] = fbfm
670
669     # write RSG data
670     out_RSG[:] = RSG
671
670     # write noise data
671     out_noise[:] = noise
672
671     # write viscosity data
672     out_viscosity[:] = viscosity
673
672     # write dampening data
673     out_dampening[:] = dampening
674
673     # write beta data
674     out_beta[:] = beta
6
```

## Code “pyBurgers.py”, Ver 3

```

35     def derivative(self,u,dx,disTy):
36
37         # signal shape information
38         n = int(u.shape[0])
39
40         if (disTy==0):
41             m = int(n/2)
42
43             # Fourier collocation method
44             h = 2*np.pi/n
45             fac = h/dx
46             k[m] = 0
47             fu = np.fft.fft(u)
48             dudx = fac*np.real(np.fft.ifft(cm.sqrt(-1)*k*fu))
49
50
51         if (disTy==4): #fourth-order fully-conservative in physical space
52             dudx = np.zeros(n) #!!! Implementation is required
53             d2udx2 = np.zeros(n) #!!! Implementation is required
54             du2dx = np.zeros(n) #!!! Implementation is required
55
56         else: #second-order fully-conservative in physical space
57             dudx = np.zeros(n) #!!! Implementation is required
58             d2udx2 = np.zeros(n) #!!! Implementation is required
59             du2dx = np.zeros(n) #!!! Implementation is required
60
61
62         # store derivatives in a dictionary for selective access
63         derivatives = {
64             'dudx' : dudx,
65             'du2dx' : du2dx,
66             #'d3udx3': d3udx3,
67             'd2udx2': d2udx2
68         }
69
70
71
72
73
74
75
76
77

```

Chap 6

By E. Amani

## Code “pyBurgers.py”, Ver 3

### Time loop

```

150     # compute subgrid terms
151     sgs = LES.subgrid(u,dudx,dx,kr)
152
153     tau = sgs["tau"]
154     coeff = sgs["coeff"]
155     if model==4:
156         kr = sgs["kr"]
157         RSG = 0.5*sgs["dttaudx"]
158
159     # compute right hand side
160     rhs = diff - conv + fbfm - RSG
161
162     # time integration
163     if t == 0:
164         # Euler for first time step
165         u_new = u + dt*rhs
166     else:
167         # 2nd-order Adams-Basforth
168         u_new = u + dt*(1.5*rhs - 0.5*rhs)
169
170     u = u_new
171     rhsp = rhs
172
173
174     # output to file every nStat time steps
175     if ((t+1)%nStat==0):
176
177         # kinetic energy
178         tke = 0.5*np.var(u)
179
180         # dissipation
181         diss_sgs = np.mean(-tau*dudx)
182
183
184
185
186

```

Chap 6

**Terms of the semi-discrete form:**  
 $\frac{\partial U}{\partial t} = \text{rhs} =$   
**diff – conv + fbfm – RSG**

By E. Amani

## Code “pyBurgers.py”, Ver 3

The screenshot shows a Jupyter Notebook interface with three tabs at the top: 'pyBurgers.py X', 'burgerslib.py X', and 'namelist.json X'. The main code cell contains Python code for a class 'burgerslib'.

```
# Fourier box filter
def filterBox(self,u,k):
    # signal size information
    n = int(u.shape[0])
    m = int(n/k)
    l = int(m/2)

    # compute fft then filter
    fu = np.fft.fft(u)
    fuf = np.zeros(n,dtype=np.complex)
    fuf[0:l] = fu[0:l]
    fuf[n-l+1:n] = fu[n-l+1:n]

    # return from spectral space
    uf = np.real(np.fft.ifft(fuf))

    return uf

# Physical space box filter, k=delta/dx
def filterBoxPS(self,u,k):
    # signal size information
    n = int(u.shape[0])

    uf = np.zeros(n) #!!!Implementation is required

    return uf
```

Chap 6

By E. Amani

## Code “pyBurgers.py”, Ver 3

```

204     # function to compute subgrid terms in Fourier space
205     def subgrid(self,u,dudx,dx,kr):
307         # Deardorff TKE
308         if self.model==4:
309             Ce = 0.78
310             C1 = 0.1
311             dt = self.dt
312
313             if (self.disTy==0):
314                 d1 = utils.dealias1(np.abs(dudx),n)
315                 d2 = utils.dealias1(dudx,n)
316                 d3 = utils.dealias2(d1*d2,n)
317             else:
318                 d3 = np.abs(dudx)*dudx
319
320             derivs_kru = utils.derivative(u*kr,dx,self.disTy)

336         # exception when none selected
337         else:
338             raise Exception("Please choose an SGS model in namelist.\n"
339                             "0=no model\n"
340                             "1=constant-coefficient Smagorinsky\n"
341                             "2=dynamic Smagorinsky\n"
342                             "3=dynamic Wong-Lilly\n"
343                             "4=Deardorff 1.5-order TKE")
344
345         derivs_tau = utils.derivative(tau,dx,self.disTy)
346         dtaudx = derivs_tau['dudx']
347
348         sgs = {
349             'tau' : tau,
350             'dtaudx' : dtaudx,
351             'coeff' : coeff,
352             'kr' : kr
353         }
354
355         return sgs

```

Chap 6

By E. Amani

