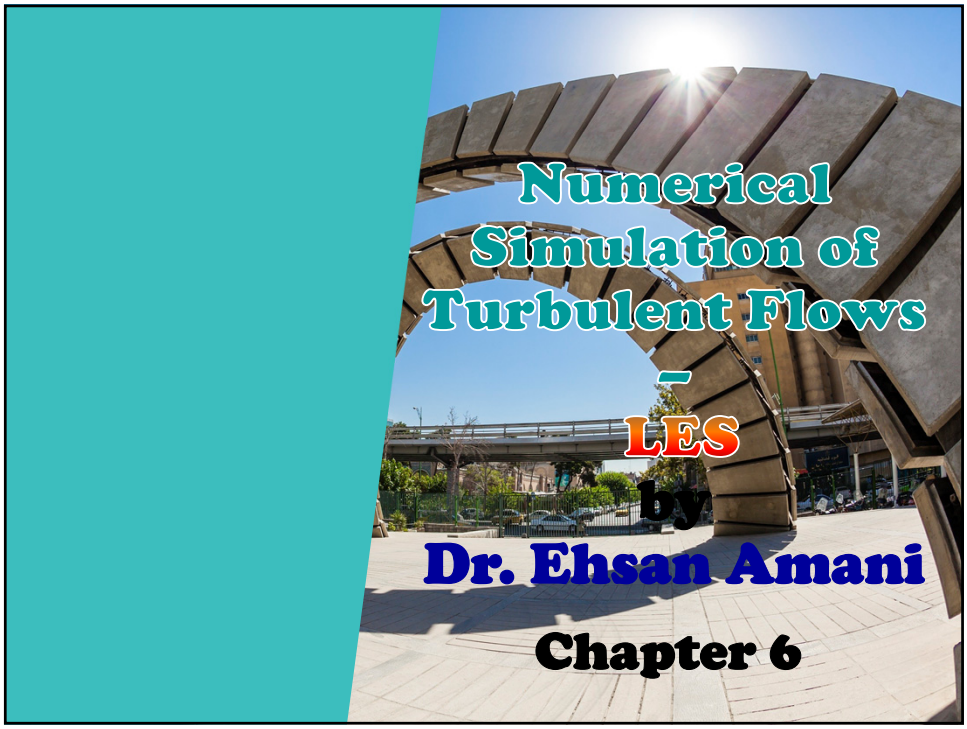


Large-Eddy Simulation (LES)

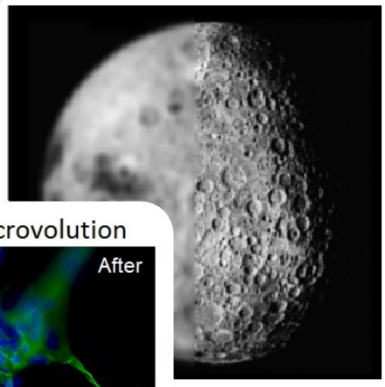
Four parts:

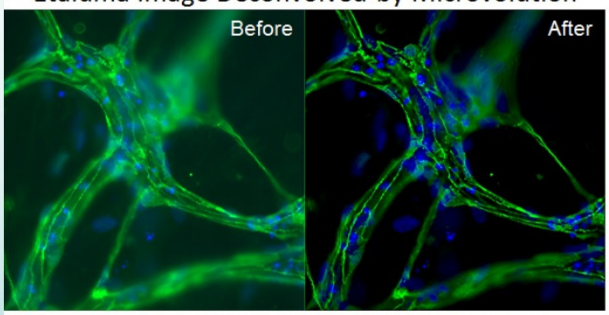
1. Filtering operation
2. Filtered equations
3. Closure: Subfilter modeling (modeling error)
4. Numerics (numerical error)

VI.1 Filtering

Applications

- Image processing
  - deconvolution





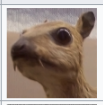
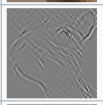
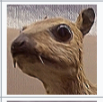
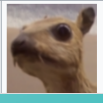
Cells in 3D channel of microfluidic chip wide-field imaged by Etaluma LS720 Microscope

Chap 6 By E. Amani

VI.1 Filtering

Applications

- Image processing
  - Deconvolution
  - ...

Operation	Kernel ω	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 × 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)

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
  - Definition:

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

$d\vec{r} = dr_1 dr_2 dr_3$

Domain

Filter function (kernel)

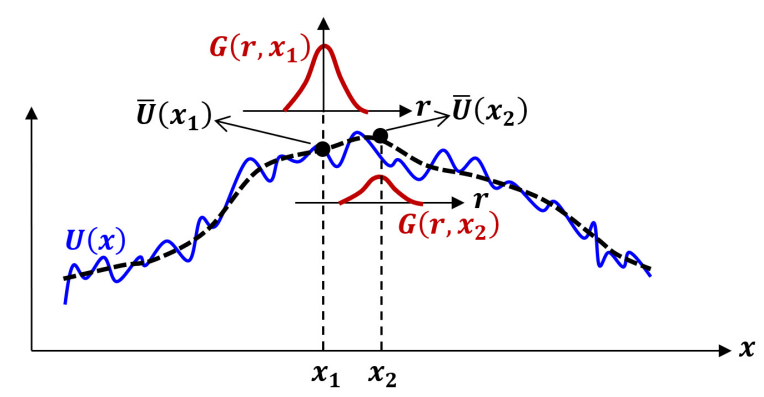
  - If $G = \delta(\vec{r}) \rightarrow \overline{\vec{U}} = \vec{U}$

Delta function

VI.1 Filtering

Convolution filters

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



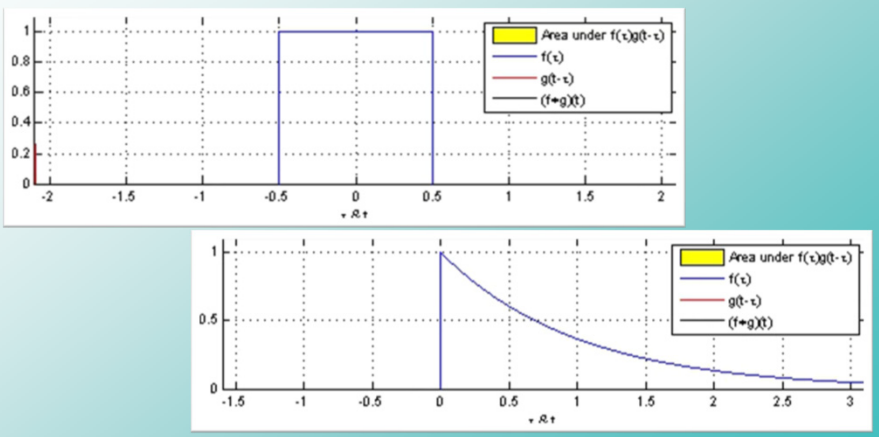
▲ Representation of the filter action, Eq. (6.1), in a 1D case.

Chap 6 By E. Amani

VI.1 Filtering

Convolution filters

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

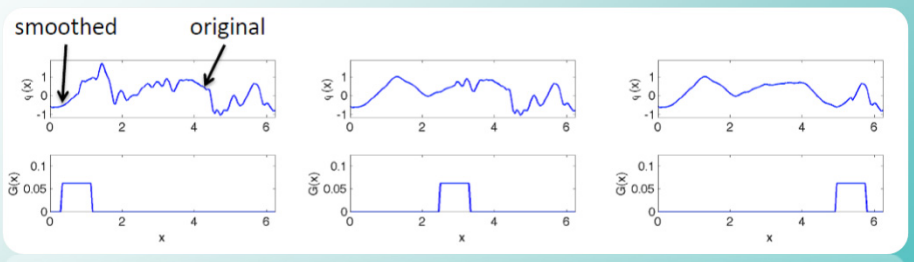


Chap 6 By E. Amani

VI.1 Filtering

Convolution filters

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

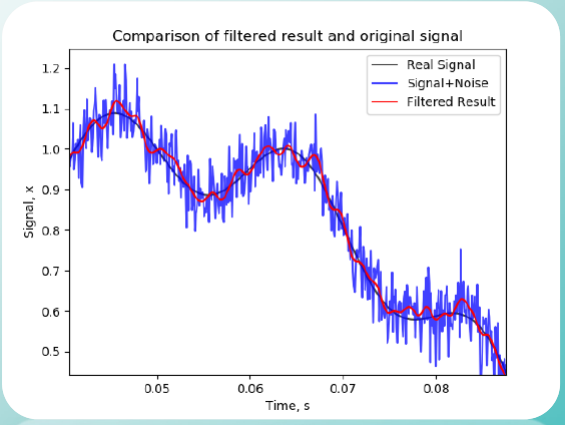


Chap 6 By E. Amani

VI.1 Filtering

Convolution filters

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

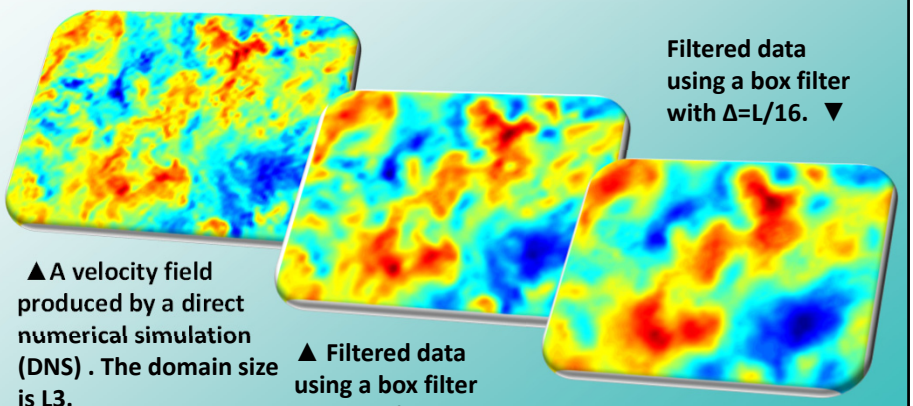


Chap 6 By E. Amani

VI.1 Filtering

Convolution filters

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



Filtered data using a box filter with Δ=L/16. ▼

▲A velocity field produced by a direct numerical simulation (DNS) . The domain size is L3.

▲ Filtered data using a box filter with Δ=L/32.

Chap 6 By E. Amani

VI.1 Filtering

General definitions and properties of filters

- The normalization condition:

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

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

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

Filtered (resolved) Subfilter (residual)

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

Chap 6 By E. Amani

VI.1 Filtering

General definitions and properties of filters

- Filtering properties:

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

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

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

- Homogeneous filters: The same kernel over all domain

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

- Homogeneous + spherically symmetric: Isotropic filter

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

VI.1 Filtering

General definitions and properties of filters

- Exercise: show that for homogeneous filters

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

- 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.

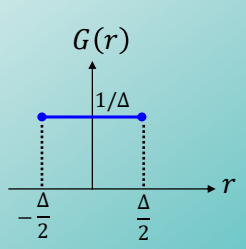
VI.1 Filtering

Filter types

- The box (top-hat) filter:
  - 1D box filter

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

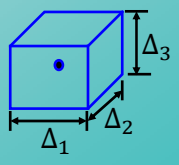
Filter width



  - 3D box filter

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

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



  - Equivalent to the volume averaging in the cubic box

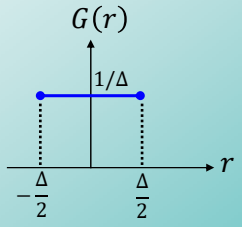
Chap 6 By E. Amani

VI.1 Filtering

Filter types

- The box (top-hat) filter:
  - 1D box filter

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



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

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

Chap 6 By E. Amani

VI.1 Filtering

Filter types

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

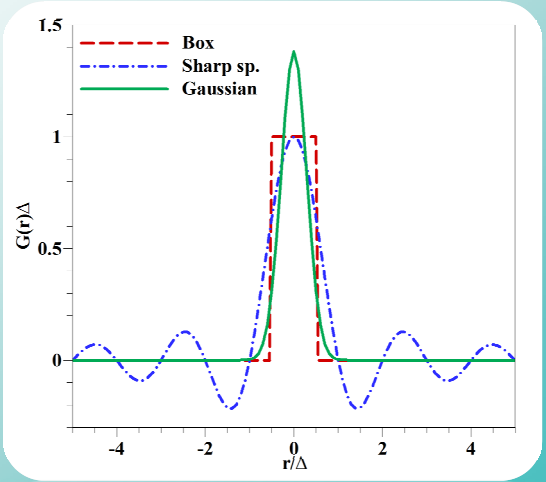
◄ Table 13.2 [1], filters formula in physical space

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

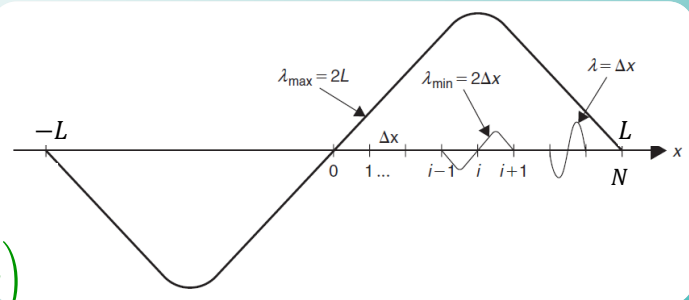
Implicit filtering – numerical diffusion

⇨ **Lecture Notes: 6.1.3**

VI.1 Filtering

Implicit filtering – numerical diffusion

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



▲ **Fourier transform of the solution**

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

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

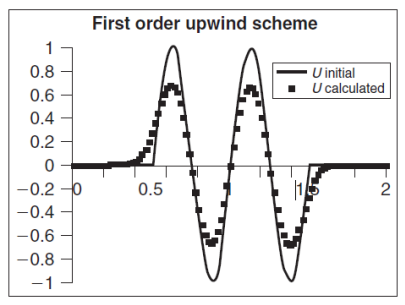
$$\phi = \frac{j\pi}{N}; j = 0, \dots, N \quad (0 < \phi < \pi)$$

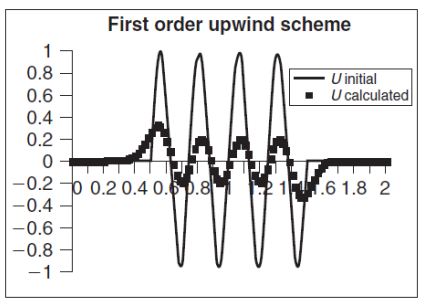
VI.1 Filtering

Implicit filtering – numerical diffusion

- Pure advection test: after 80 time steps

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





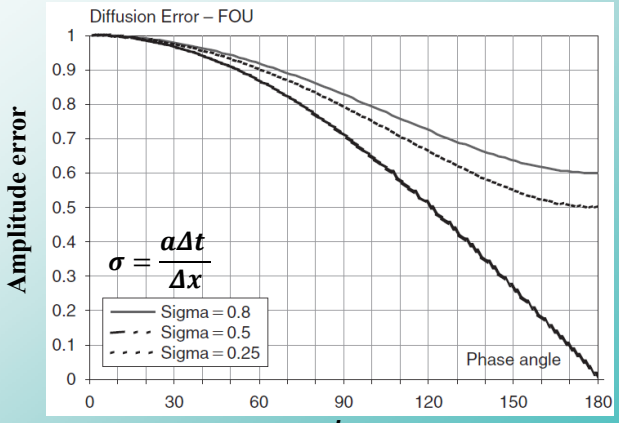
$\phi = \frac{j\pi}{N} = \frac{8\pi}{100}$ ▲ ▲ $\phi = \frac{j\pi}{N} = \frac{16\pi}{100}$

Chap 6 By E. Amani

VI.1 Filtering

Implicit filtering – numerical diffusion

- Pure advection test:



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

Chap 6 By E. Amani

VI.1 Filtering

Implicit filtering – numerical diffusion

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

VI.2 Filtered equations

Taking filter of the Navier-Stokes equations:

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

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

- The term $\overline{U_i U_j}$ raises the closure problem

$$\overline{U_i U_j} \equiv \bar{U}_i \bar{U}_j + \tau_{ij}^R \qquad \tau_{ij}^R \equiv \overline{U_i U_j} - \bar{U}_i \bar{U}_j \qquad (6.30)$$

→ **Residual or SubGrid-Scale (SGS) stress**

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

- 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)?

VI.2 Filtered equations

The filtered fluctuation:

- Remember

Resolved part of fluctuation

Similarly

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

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

Chap 6 By E. Amani

VI.2 Filtered equations

Decomposition of kinetic energies:

- Turbulent kinetic energy:

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

filtered subfilter

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

resolved Unresolved (subfilter-scale)

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

residual

Chap 6 By E. Amani

VI.2 Filtered equations

Decomposition of kinetic energies:

- (Instantaneous) kinetic energy:

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

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

E^R or E_f

$$\bar{E} = E^R + E_r \qquad E_r = \underbrace{\frac{1}{2}(\overline{U_i U_i} - \bar{U}_i\bar{U}_i)}_{\frac{1}{2}\tau_{ii}^R} \qquad (6.51)' \ (E^{\mathrm{SFS}} = E' + E_r)$$

E_r or K_r residual

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

VI.2 Filtered equations

Decomposition of kinetic energies:

- Exercise: show that

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

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

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

- Transport equation for E^R [1]:

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

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

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

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

VI.2 Filtered equations

Decomposition of kinetic energies:

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

$$\frac{\bar{D}E_r}{\bar{D}t} + \frac{\partial}{\partial x_j}\left[\frac{1}{2}\overline{U_i U_i U_j} - \frac{1}{2}\overline{U_i U_i}\bar{U}_j + \frac{1}{\rho}\overline{pU_j} - \frac{1}{\rho}\bar{p}\bar{U}_j - \bar{U}_i\tau_{ij}^r\right] - \qquad (6.60)''$$

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

$$\varepsilon_r = \nu\left(\overline{\frac{\partial U_i}{\partial x_j}\frac{\partial U_i}{\partial x_j}} - \overline{\frac{\partial U_i}{\partial x_j}}\,\overline{\frac{\partial U_i}{\partial x_j}}\right) \qquad (6.91)''$$

VI.2 Filtered equations

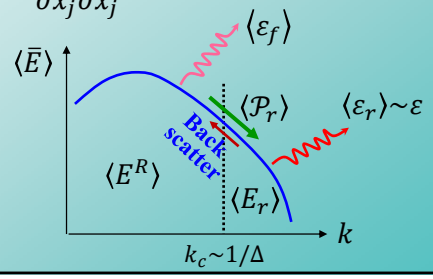
Decomposition of kinetic energies:

- Energy cascade:

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

$$\frac{\bar{D}E_r}{\bar{D}t} + \frac{\partial}{\partial x_j}\left[\frac{1}{2}\overline{U_i U_i U_j} - \frac{1}{2}\overline{U_i U_i}\bar{U}_j + \frac{1}{\rho}\overline{pU_j} - \frac{1}{\rho}\bar{p}\bar{U}_j - \bar{U}_i\tau_{ij}^r\right] - \qquad (6.60)''$$

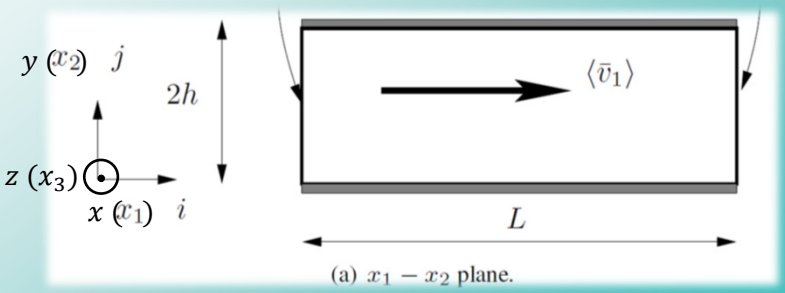
$$\nu\frac{\partial^2 E_r}{\partial x_j \partial x_j} = +\mathcal{P}_r - \varepsilon_r$$



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!)

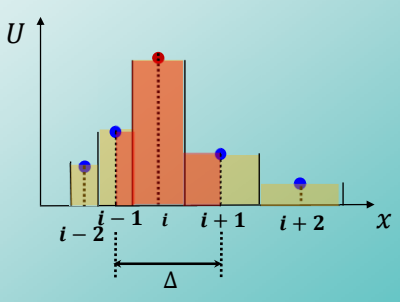


Chap 6 By E. Amani

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

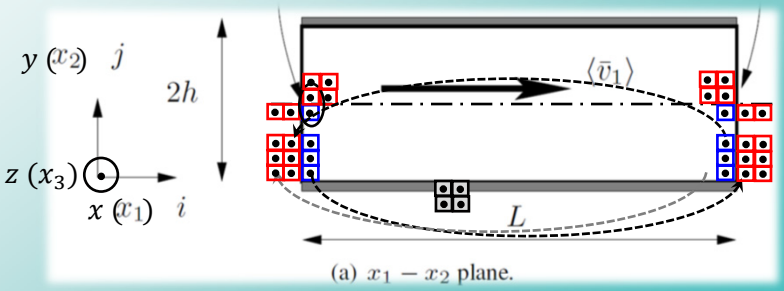


Chap 6 By E. Amani

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!



(a) $x_1 - x_2$ plane.

Chap 6 By E. Amani

VI.3 SubGrid-Scale (SGS) modeling

The Smagorinsky model

- The eddy-viscosity assumption
- Remember:

$$\langle u_i u_j \rangle - \underbrace{\frac{2}{3}k}_{\frac{1}{3}\langle u_k u_k \rangle}\delta_{ij} = -2\nu_T \langle S_{ij} \rangle \qquad (5.11)$$

- Similarly,

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

↘ SGS eddy viscosity

$$\bar{S}_{ij} = \frac{1}{2}\left(\frac{\overline{\partial U_i}}{\partial x_j} + \frac{\overline{\partial U_j}}{\partial x_i}\right) \qquad (6.41)$$

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

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

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

- ν_r =?
- The kinetic theory of gasses for the molecular viscosity:

$$\nu = v\ell$$

Standard deviation of molecular velocity — **Mean free path**

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

$$\nu_t = v_m \ell_m \qquad (6.43)$$

VI.3 SubGrid-Scale (SGS) modeling

The Smagorinsky model

- A simple estimate for v_m and ℓ_m in free shear flows:

⇨ **Lecture Notes: 6.3.1**

VI.3 SubGrid-Scale (SGS) modeling

The Smagorinsky model

- A simple estimate for ν_m and ℓ_m in free shear flows:

$$\nu_t = \ell_m \nu_m = \ell_m^2 \left| \frac{d\langle U \rangle}{dy} \right| \qquad (6.44)$$

- Smagorinsky's relation for ν_r:

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

C_s: Smagorinsky constant

VI.3 SubGrid-Scale (SGS) modeling

Tests of model performances

- *A priori* test: Using DNS data

$$\tau_{ij}^r \equiv \overline{U_i U_j} - \bar{U}_i \bar{U}_j - \frac{1}{3}[\overline{U_k U_k} - \bar{U}_k \bar{U}_k]\delta_{ij} \Leftrightarrow 2(C_s\Delta)^2 \bar{S}\bar{S}_{ij} = \tau_{ij}^{r,M}$$

$\overline{U_i U_j}$ ← Filtering $U_i U_j$ ← U_i → Filtering $\bar{U}_i$ → $\bar{S}_{ij}$ → $\bar{S}$

U_i → DNS

  - The correlation coefficient is usually used as a measure

- *A posteriori* test: LES

DNS → U_i → (Filtering) $\bar{U}_i$ → (Averaging) $\langle \bar{U}_i \rangle$ ⇔ $\langle \bar{U}_i \rangle$ ← (Averaging) $\bar{U}_i$ ← LES

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 ($C_s \nrightarrow 0$)
- A possible remedy: Van Driest damping function

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

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

VI.3 SubGrid-Scale (SGS) modeling

The dynamic Smagorinsky model

⇨ **Lecture Notes: 6.3.3**

VI.3 SubGrid-Scale (SGS) modeling

Alternative SGS modeling

- Remember:

$$\overline{U_iU_j} \equiv \underbrace{\overline{U}_i\overline{U}_j}_{\text{Resolved part}} + \underbrace{\tau_{ij}^R}_{\text{Unresolved part}} \qquad (6.30)$$

Resolved part — Unresolved part ⟶ Modeling

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

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

Leonard stress — Cross stress — SGS Reynolds stress

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

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

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

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 τ_{ij}^R
- Exercise: Show that for $\widetilde{\Delta} = \overline{\Delta}$, the Leonard stress is the same as the resolved SDS stress.
- Using Germano's decomposition:

$$\overline{U_iU_j} \equiv \underbrace{\overline{U}_i\overline{U}_j + \mathcal{L}_{ij}^0}_{\text{Resolved}} + \underbrace{\mathcal{C}_{ij}^0 + \mathcal{R}_{ij}^0}_{\tau_{ij}^k\ \text{(Unresolved)}} \qquad (6.61)$$

Resolved — τ_{ij}^k (Unresolved) ⟶ Modeling

- Using the dynamic Smgorinsky model for τ_{ij}^k results in:

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

Chap 6 By E. Amani

VI.3 SubGrid-Scale (SGS) modeling

Alternative SGS modeling

- The dynamic Bardina model:

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

- Exercise: It can be shown that:

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

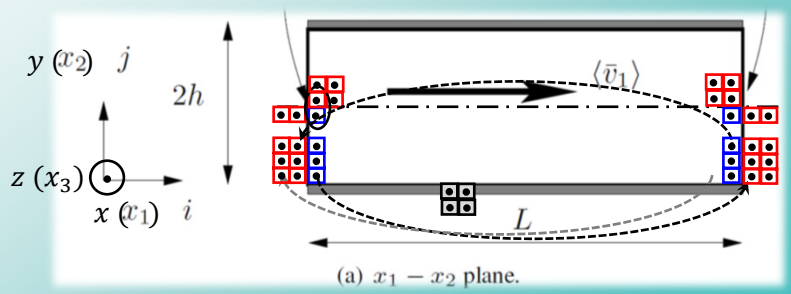
$$l_{ij}^{B} = l_{ij} - H_{ij} \qquad (6.63)$$

$$H_{ij} = \widetilde{\bar{\bar{U}}_{i}\bar{\bar{U}}_{j}} - \tilde{\bar{\bar{U}}}_{i}\tilde{\bar{\bar{U}}}_{j} \underbrace{- \left(\widetilde{\bar{U}_{i}\bar{U}_{j}} - \widetilde{\bar{U}}_{i}\widetilde{\bar{U}}_{j}\right)}_{\widetilde{\mathcal{L}_{ij}^{0}}} \qquad (6.63)'$$

Project#1: *A priori* study

Part II: SGS modeling

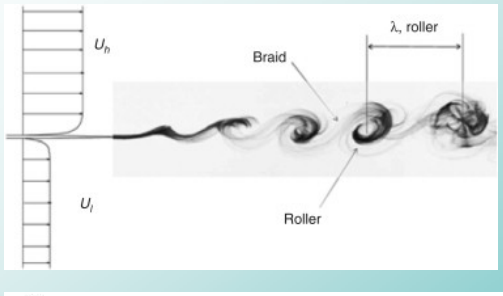
- Evaluate the performance static and dynamic SGS models
- Understanding energy cascade in a LES
- Back scatter
- The quality criteria of LES

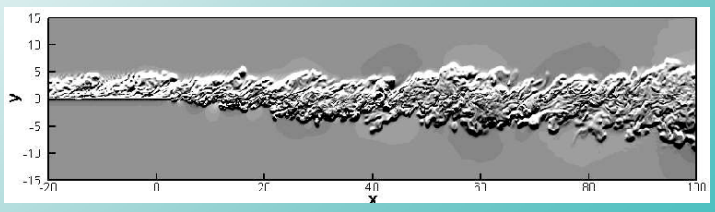


(a) $x_1 - x_2$ plane.

VI.3 SubGrid-Scale (SGS) modeling

Case study: Turbulent mixing layer





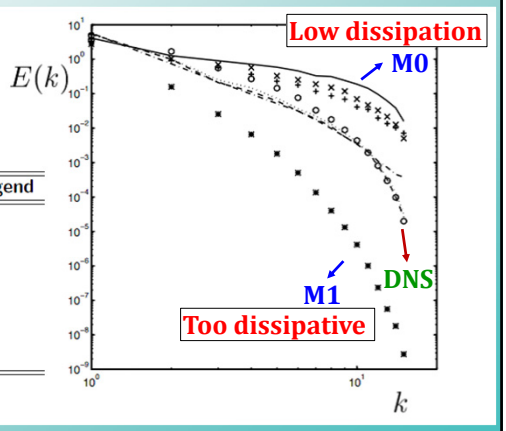
Chap 6 By E. Amani

VI.3 SubGrid-Scale (SGS) modeling

Case study: Turbulent mixing layer

An example from Geurts (2004)

Name	Model for τ_{ij}	Plot legend
M0	No model	–
M1	Smagorinsky	⋆
M2	Similarity	×
M3	Nonlinear	+
M4	Dynamic Smagorinsky	– –
M5	Dynamic Mixed	...
M6	Dynamic Nonlinear	–·



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

Chap 6 By E. Amani

VI.3 SubGrid-Scale (SGS) modeling

Beyond eddy-viscosity models

- **Approximate Deconvolution Models (ADM)**
- **… (Exercise)**

VI.4 Numerics of LES in physical space

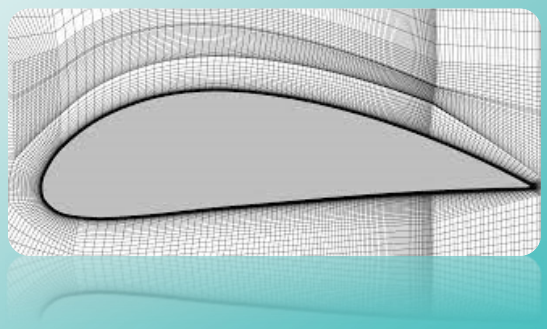
VI.4.1 Overview of CFD

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

VI.4 Numerics of LES in physical space

VI.4.1 Overview of CFD

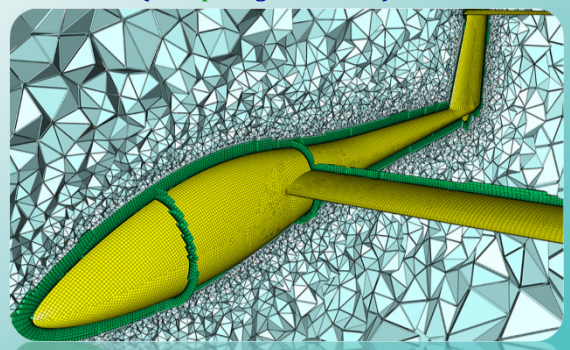
- **Steps**
  1) Computational grid:
     - Structured (simple geometries)



VI.4 Numerics of LES in physical space

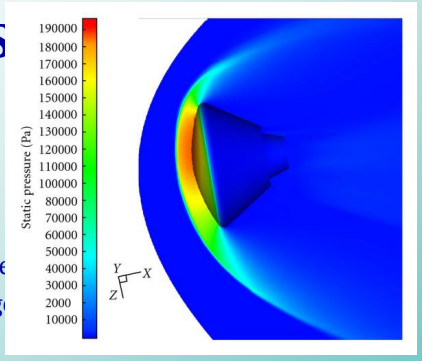
VI.4.1 Overview of CFD

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



VI.4 Numerics of LES

VI.4.1 Overview of CFD



- **Steps**
  1) Computational grid:
     - Structured (simple geome
     - Unstructured (Complex g
  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

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

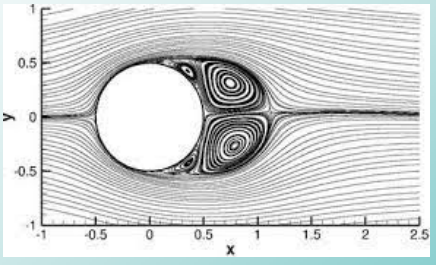
VI.4.1 Overview of CFD

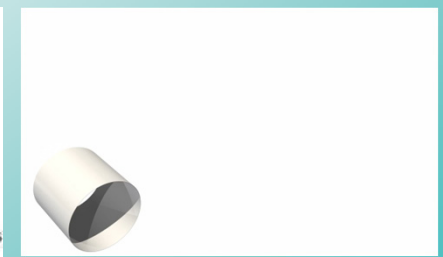
- **Steps**
  1) Computational grid:
     - Structured (simple geometries)
     - Unstructured (Complex geometries)
  2) Solution algorithm:
     - Density-based (High-Mach-number flows)
     - Pressure-based (Low-Mach-number flows)
  3) Discretization method:
  4) Treating boundary conditions
  5) Numerical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

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)

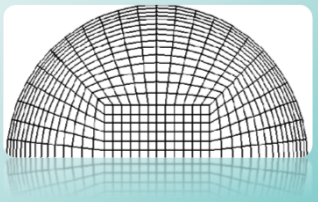


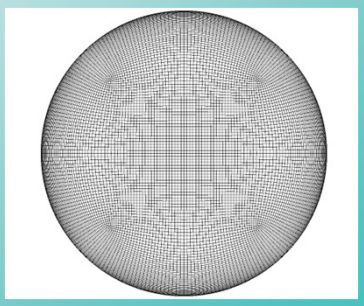


VI.4 Numerics of LES in physical space

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





VI.4 Numerics of LES in physical space

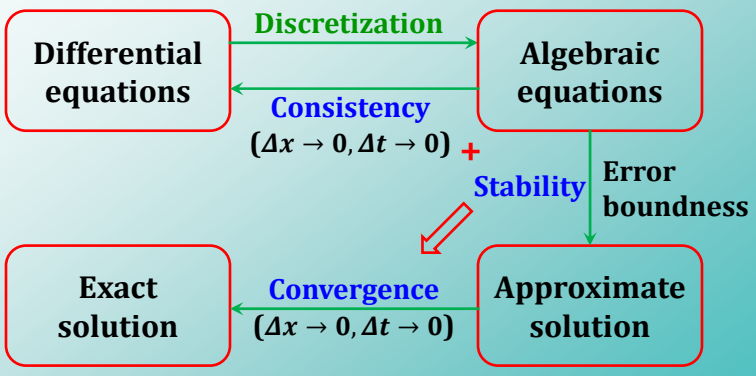
VI.4.2 CFD considerations – LES vs. RANS

- **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 2nd 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

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

    Diff. Eq. = Disc. Eq. + truncation error $(O(\Delta x^n, \Delta t^m))$

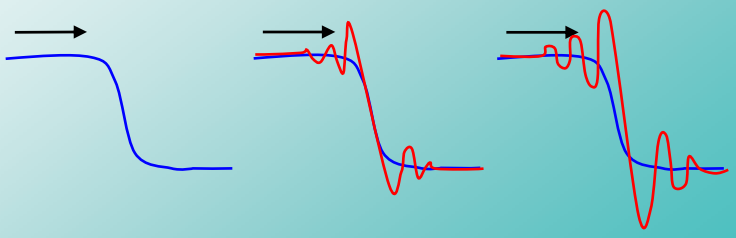
  - FV is limited to $n = 2$
  - FD and SM (simple geometries) and FE (complex geometries) are preferable for LES

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. = $[1 - \Phi(U)] \times$ **FOU** + $\Phi(U) \times$ **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

2nd-order fully-conservative scheme on a regular grid

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

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

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

$$(\text{Div.} - R2)_i \equiv \delta_{1x_j} \left(\overline{U}_j^{1x_j} \overline{U}_i^{1x_j} \right) \qquad (6.101)$$

4nd-order fully-conservative scheme on a regular grid

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

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

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

$$(\text{Div.} - R4)_i \equiv \tfrac{4}{3}\delta_{1x_j} \left(\overline{U}_j^{1x_j} \overline{U}_i^{1x_j} \right) - \tfrac{1}{3}\delta_{2x_j} \left(\overline{U}_j^{2x_j} \overline{U}_i^{2x_j} \right) \qquad (6.105)$$

Chap 6 By E. Amani

Project#2: *A posteriori* study

1D Stochastic Forced Burgers Equation (1DFSBE)

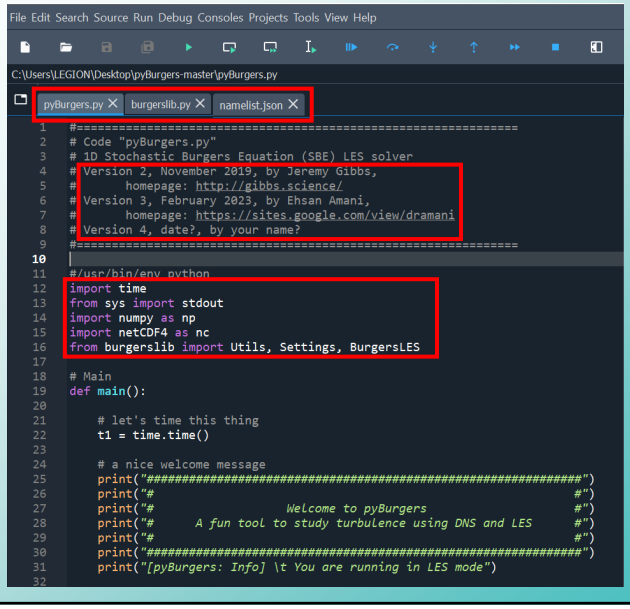
- Evaluate the performance static and dynamic SGS models
- Understanding energy cascade in a LES
- Back scatter
- The quality criteria of LES
- Energy spectrum (Q11 after section VII.1)
- "pyBurgers.py" (ver. 3) code amendment (OOP)

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

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

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 | burgerslib.py | namelist.json
```

```
#============================================================
# Code "pyBurgers.py"
# 1D Stochastic Burgers Equation (SBE) LES solver
# Version 2, November 2019, by Jeremy Gibbs,
#       homepage: http://gibbs.science/
# Version 3, February 2023, by Ehsan Amani,
#       homepage: https://sites.google.com/view/dramani
# Version 4, date?, by your name?
#============================================================

#/usr/bin/env python
import time
from sys import stdout
import numpy as np
import netCDF4 as nc
from burgerslib import Utils, Settings, BurgersLES

# Main
def main():

    # let's time this thing
    t1 = time.time()

    # a nice welcome message
    print("#################################################################")
    print("#                                                               #")
    print("#                  Welcome to pyBurgers                         #")
    print("#      A fun tool to study turbulence using DNS and LES         #")
    print("#                                                               #")
    print("#################################################################")
    print("[pyBurgers: Info] \t You are running in LES mode")
```

Chap 6 By E. Amani

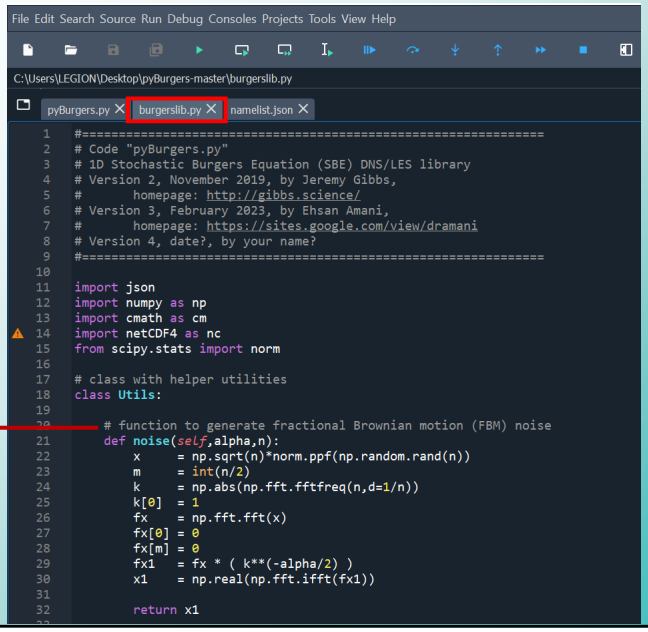
Code "pyBurgers.py", Ver 3

burgerslib

```
    # instantiate helper classes
    print("[pyBurgers: Setup] \t Reading input settings")
    utils    = Utils()
    settings = Settings('nameList.json')

    # input settings
    nxDNS = settings.nxDNS
    nxLES = settings.nxLES
    mp    = int(nxLES/2)
    dx    = 2*np.pi/nxLES
    dt    = settings.dt
    nt    = settings.nt
    model = settings.sgs
    visc  = settings.visc
    damp  = settings.damp
    beta = settings.beta
    nInfo = settings.nInfo
    nStat = settings.nStat
    disTy = settings.disTy

    # intantiate the LES SGS class
    LES = BurgersLES(model)

    # initialize velocity field
    print("[pyBurgers: Setup] \t Initialzing velocity field")
    u = np.zeros(nxLES)

    # initialize subgrid tke if using Deardorff
    if model==4:
        kr = np.ones(nxLES)
    else:
        kr = 0

    # initialize random number generator
    np.random.seed(1)
```

Chap 6 By E. Amani

Code "pyBurgers.py", Ver 3

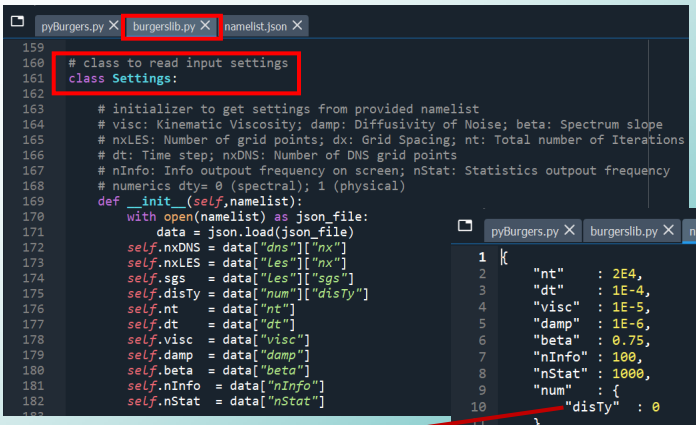


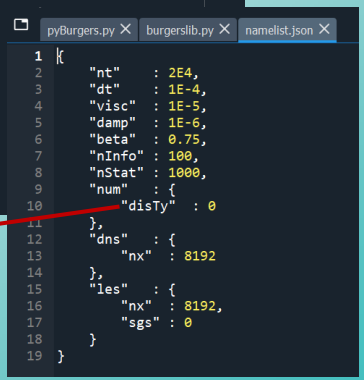
Forcing function

Chap 6

By E. Amani

Code "pyBurgers.py", Ver 3





0: Spectral

2: 2nd-order fully-conservative

4: 4nd-order fully-conservative

Chap 6

By E. Amani

Code "pyBurgers.py", Ver 3

pyBurgers.py | burgerslib.py | namelist.json

```
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,kr):
206
207         # signal size information
208         n = int(u.shape[0])
```

No SGS stress

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     # intantiate the LES SGS class
54     LES = BurgersLES(model)
55
56     # initialize velocity field
57     print("[pyBurgers: Setup] \t Initialzing 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)
```

Accessing object data

Accessing object functions

Chap 6 By E. Amani

Code "pyBurgers.py", Ver 3

Initialization, output files, and grid

pyBurgers.py | burgerslib.py | namelist.json

```
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)
68
69      # place holder for right hand side
70      rhsp = 0
71      diff = 0
72      conv = 0
73      fbmf = 0
74      RSG = 0
75
76      # create output file
77      print("[pyBurgers: Setup] \t Creating output file")
78      output = nc.Dataset('pyBurgersLES.nc','w')
```

```
120     # write x data
121     out_x[:] = np.arange(0,2*np.pi,dx)
122
```

Terms of the semi-discrete form:

$$\frac{\partial U}{\partial t} = \text{rhs} = \text{diff} - \text{conv} + \text{fbmf} - \text{RSG}$$

Code "pyBurgers.py", Ver 3

Time loop

pyBurgers.py | burgerslib.py | namelist.json

```
123     # time loop
124     save_t = 0
125     for t in range(int(nt)):
126
127         # update progress
128         if (t==0 or (t+1)%nInfo==0):
129             stdout.write("\r[pyBurgers: LES] \t Running for time %07d of %d"%(t+1,int(nt)))
130             stdout.flush()
131
132         # add fractional Brownian motion (FBM) noise
133         fbm  = np.sqrt(2*damp/dt)*utils.noise(beta,nxDNS)
134
135         #filtering on the DNS grid and then transfering to the LES grid
136         if (disTy==0):
137             if (nxDNS>nxLES): fbmf = utils.filterDown(fbm,int(nxDNS/nxLES))
138         else:
139             if (nxDNS>nxLES): fbmf = utils.filterBoxPS(fbm,int(nxDNS/nxLES))[::int(nxDNS/nxLES)]
140
141         # compute derivatives
142         derivs = utils.derivative(u.dx.disTy)
143         dudx   = derivs['dudx']
144         du2dx  = derivs['du2dx']
145         d2udx2 = derivs['d2udx2']
146         #d3udx3 = derivs['d3udx3']
147         conv  = 0.5*du2dx
148         diff = visc*d2udx2
```

Sharp-spectral filter

Box filter

Code "pyBurgers.py", Ver 3

pyBurgers.py | burgerslib.py | namelist.json

```
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 colocation method
44              h       = 2*np.pi/n
45              fac     = h/dx
46              k       = np.fft.fftfreq(n,d=1/n)
47              k[m]    = 0
48              fu      = np.fft.fft(u)
49              dudx    = fac*np.real(np.fft.ifft(cm.sqrt(-1)*k*fu))
```

```
62          if (disTy==4): #fourth-order fully-conservative in physical space
63              dudx  = np.zeros(n) #!!! Implementation is required
64              d2udx2  = np.zeros(n) #!!! Implementation is required
65              du2dx   = np.zeros(n) #!!! Implementation is required
66          else: #second-order fully-conservative in physical space
67              dudx  = np.zeros(n) #!!! Implementation is required
68              d2udx2  = np.zeros(n) #!!! Implementation is required
69              du2dx   = np.zeros(n) #!!! Implementation is required
70
71          # store derivatives in a dictionary for selective access
72          derivatives = {
73              'dudx'  :   dudx,
74              'du2dx' :   du2dx,
75              #'d3udx3':   d3udx3,
76              'd2udx2':   d2udx2
77          }
```


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["dtaudx"]
158
159         # compute right hand side
160         rhs = diff - conv + fbmf - 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-Bashforth
168             u_new = u + dt*(1.5*rhs - 0.5*rhsp)
```

```
176         u           = u_new
177         rhsp        = rhs
178
179         # output to file every nStat time steps
180         if ((t+1)%nStat==0):
181
182             # kinetic energy
183             tke  = 0.5*np.var(u)
184
185             # dissipation
186             diss_sgs = np.mean(-tau*dudx)
```

Terms of the semi-discrete form:

$$\frac{\partial U}{\partial t} = \text{rhs} = \text{diff} - \text{conv} + \text{fbmf} - \text{RSG}$$

Code "pyBurgers.py", Ver 3

pyBurgers.py | burgerslib.py | namelist.json

```
100     # Fourier box filter
101     def filterBox(self,u,k):
102
103         # signal size information
104         n   = int(u.shape[0])
105         m   = int(n/k)
106         l   = int(m/2)
107
108         # compute fft then filter
109         fu  = np.fft.fft(u)
110         fuf = np.zeros(n,dtype=np.complex)
111         fuf[0:l]     = fu[0:l]
112         fuf[n-l+1:n] = fu[n-l+1:n]
113
114         # return from spectral space
115         uf = np.real(np.fft.ifft(fuf))
116
117         return uf
118
119     # Physical space box filter, k=delta/dx
120     def filterBoxPS(self,u,k):
121
122         # signal size information
123         n   = int(u.shape[0])
124
125         uf = np.zeros(n) #!!!Implementation is required
126
127         return uf
```


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 = 0.70
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         return sgs
```