pyCALC-LES

www.tfd.chalmers.se

Lars Davidson, M2 Fluid Dynamics
Chalmers University of Technology
Gothenburg, Sweden

SCHEDULE, DAY 1

- Lecture
  - Presentation of **pyCALC-LES**
- Workshop
  - Working with **pyCALC-LES**
- Lecture
  - LES & DES eBook
- Workshop
  - Working with **pyCALC-LES**

SCHEDULE, DAY 2

- Lecture
  - PANS & synthetic inlet b.c. eBook
- Workshop
  - Working with **pyCALC-LES**
  - Individual break-out Zoom meetings
- Lecture
  - Machine Learning: Neural network and EARSM
  - `slides-EARSM-slides-flow-induced-acoustic-seminar-huadong.pdf`
  - Paper can be downloaded here
- Workshop
  - Assignment on NN for improving a $k-\varepsilon$ model can be found here
  - Python scripts for EARSM can be downloaded here
  - Assignment on NN for improving EARSM in recirculating flow can be found here
  - Working with Neural Network
  - Individual break-out Zoom meetings
  - Working with **pyCALC-LES** , **pyCALC-RANS** or rans-k-eps-NN.py

SCHEDULE, DAY 3

- Lecture
  - PINN (Physics-Informed Neural Network)
  - `slides-machine-learning-k-omega-PINN-pinn.pdf`
  - Paper can be downloaded here
- Workshop
  - Working with PINN and **pyCALC-RANS**
  - Python scrips can be downloaded here
  - Individual break-out Zoom meetings
- Lecture
  - Machine Learning: improving wall functions. Download slides
- Workshop
  - Working with KDTree. Folder:
    `channel-16000-IDDES-wall-functions-nj98-kdtree-mean-from-backflow-A`
  - Section 29 in eBook
  - Individual break-out Zoom meetings

pyCALC-LES flowchart ▸ Link

There are four main case-specific routines in each subdirectory

- `generate*.py` (`generate-channel-grid.py`, `generate-bound-layer-grid.py`, ...)
- `setup_case.py`
- `modify_case.py`
- `pl*.py` (`pl_uvw.py`, `plot_inlet.py`, ...)

The main program

- `pyCALC-LES-GPY.py`
- `synt_fluct.py`

- Generate simple Cartesian meshes

SECTION 0 choice of CPU or GPU
SECTION 1 choice of differencing scheme
SECTION 2 turbulence models
SECTION 3 restart/save
SECTION 4 fluid properties
SECTION 5 relaxation factors
SECTION 6 number of iteration and convergence criteria
SECTION 7 all variables are printed during the iteration at node
SECTION 8 time-averaging
SECTION 9 residual scaling parameters
SECTION 10 boundary conditions

MODIFY_CASE.PY

```
def modify_init(u3d,v3d,w3d,k3d,om3d,eps3d,vis3d,dist3d):
def modify_inlet():
def modify_conv(convw,convs,convl):
def modify_u(su3d,sp3d):
def modify_v(su3d,sp3d):
def modify_w(su3d,sp3d):
def modify_k(su3d,sp3d,gen):
def modify_eps(su3d,sp3d):
def modify_om(su3d,sp3d,comm_term):
def modify_outlet(convw):
def modify_fk(fk3d):
def modify_vis(vis3d):
def def fix_k():
def def fix_omega():
def def fix_eps():
```

CREATE EXECUTABLE

The bash script `run-python` is used which reads

```
#!/bin/bash # delete first line
sed '/setup_case()/d' setup_case.py > temp_file
# add new first line plus global declarations
cat ../global-GPU temp_file modify_case.py ../STG-GPU.py
../synt_fluct-GPU.py ../pyCALC-LES-GPU.py > temp_file1;
# find setting of 'gpu'
'grep' gpu setup_case.py > gpu_value;
# remove leading white space
sed -i 's/^[ ]* //' gpu_value;
# add value of 'gpu' at first line
cat gpu_value temp_file1 > exec-pyCALC-LES-GPU.py;
/anaconda3/bin/python -u exec-pyCALC-LES-GPU.py > out
```

GLOBAL DECLARATIONS

They are made in the file `global-GPU` which reads

global acrank,acrank_conv, acrank_conv_keps, acrank_conv_kom,
. . . w_bc_south, w_bc_south_type, w_bc_west, w_bc_west_type,
w_bc_z, w_bc_z_type, wale, x2d, xp2d, y2d, yp2d, zmax

Fully-Developed Channel Flow, $Re_\tau = 500$, DNS

- `generate-channel-grid.py`
- `setup_case.py`
- `modify_case.py`

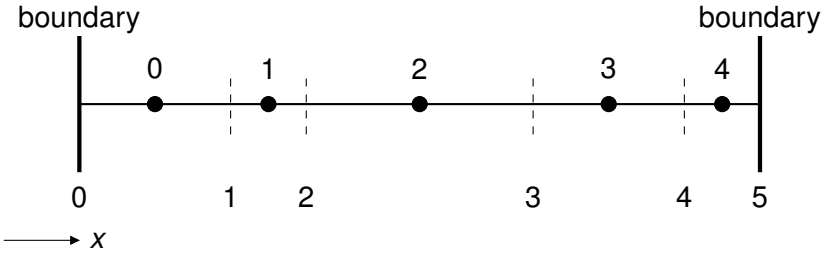


FIGURE: 1D grid. `ni=5`. The bullets denote cell centers (and control volume) which are labeled 0–4. Dashed lines denote control volume faces labeled 0–5.

- The variables are stored in cells 0 – 4, e.g. $u[0]$ – $u[4]$.
- No dummy nodes, i.e. no $u[-1]$ and $u[5]$.
- Boundary conditions are introduced as source terms

GRID: WEST, EAST, SOUTH, NORTH ($x - y$)

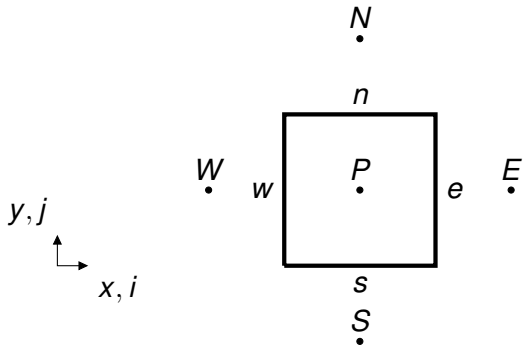
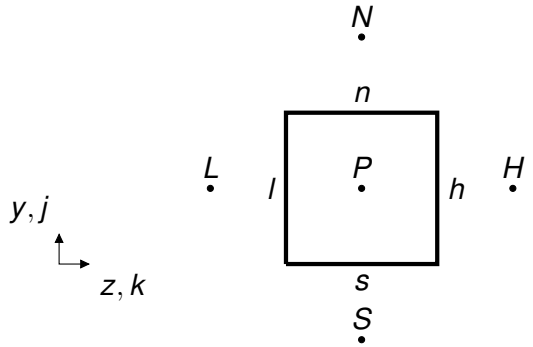


FIGURE: Control volume. Top: $x - y$ plane; bottom: $y - z$ plane.

- Interpolation

$$\phi_w = f_x \phi_P + (1 - f_x)\phi_W, \quad f_x = \frac{|\overrightarrow{Ww}|}{|\overrightarrow{Pw}| + |\overrightarrow{Ww}|}$$

GRID: SOUTH, NORTH, LOW, HIGH ($y - z$)



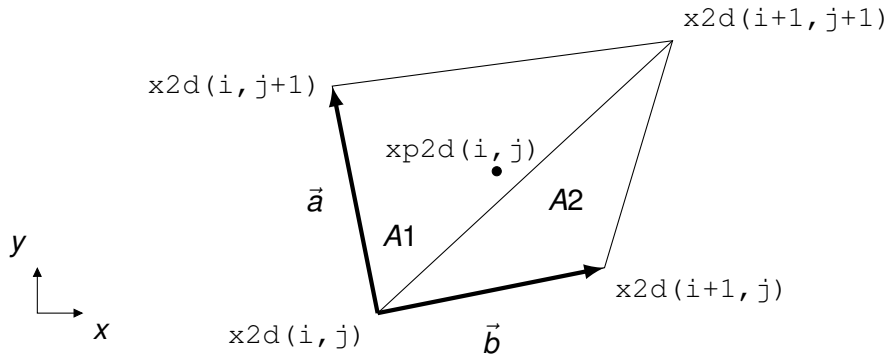


FIGURE: Calculation of areas and volume of cell `i,j,k`.

- The derivatives of ϕ ($\partial\phi/\partial x_i$) in a cell:

$$\frac{\partial \Phi}{\partial x} = \frac{1}{V}\int_A \Phi n_x dA, \quad \frac{\partial \Phi}{\partial y} = \frac{1}{V}\int_A \Phi n_y dA, \quad \frac{\partial \Phi}{\partial z} = \frac{1}{V}\int_A \Phi n_z dA$$

- For the x component, for example, we get

$$\frac{\partial \Phi}{\partial x} = \frac{1}{V}\left(\Phi_e A_{ex} - \Phi_w A_{wx} + \Phi_n A_{nx} - \Phi_s A_{sx} + \Phi_h A_{hx} - \Phi_l A_{lx}\right)$$

- The derivative $\partial\Phi/\partial x$, $\partial\Phi/\partial x$ and $\partial\Phi/\partial z$, are computed in the Python modules `dphidx`, `dphidy` and `dphidz`, respectively.

DISCRETIZED EQUATIONS

- They all read (u, v, w, k, ε, ω)

$$
\begin{aligned}
a_P\phi_P &= a_E\phi_E + a_W\phi_W + a_N\phi_N + a_S\phi_S + a_H\phi_H + a_L\phi_L + S_U \\
a_P &= a_E + a_W + a_N + a_S + a_H + a_L - S_P
\end{aligned}
$$

  - Arrays: `aw3d`, `ae3d`, ..., `ah3d`, `su3d`, `sp3d`
  - $S = S_P\phi + S_U$
  - Three different discretization schemes: central differencing scheme (CDS), 1st-order upwind, hybrid 1st-order upwind/CDS
  - This is done in `coeff()`

- compute convection $\mathbf{u} \cdot \mathbf{A}$

```
convw,convs,convl=conv(u3d,v3d,w3d,p3d_face_w, ...
```

- boundary conditions

```
su3d,sp3d=bc(su3d,sp3d,u_bc_west,u_bc_east,u_bc_south ...
```

- Compute coefficients $a_W, \ldots a_H$

```
aw3d,ae3d,as3d,an3d,al3d,ah3d,apo3d,su3d,sp3d=coeff(convw,convs ...
```

- Add generic source, i.e. $\partial \bar{p}/\partial x$

```
su3d,sp3d=calcu(su3d,sp3d,dpdx_old,p3d_face_w,p3d_face_s))
```

- Add Crank-Nicolson

```
ap3d,su3d=crank_nicol(u3d_old,aw3d,ae3d,as3d,an3d,al3d, ...
```

- Solve $\bar{u}$

```
if solver_vel == 'pyamg':
  u3d,residual_u=solve_pyamg(u3d,aw3d,ae3d,as3d,an3d,al3d,ah3d, ...
elif solver_vel == 'pyamgx':
  u3d,residual_u=solve_pyamgx(u3d,aw3d,ae3d,as3d,an3d,al3d, ...
else:
# cgs, cg, gmres, qmr, lgmres
  u3d,residual_u=solve_3d(u3d,aw3d,ae3d,as3d, ...)
```

FLOW CHART FOR $\bar{p}$/CONTINUITY EQ. IN MAIN

- Compute convection with new $\bar{u}$, $\bar{v}$, $\bar{w}$

```
convw,convs,convl=conv(u3d,v3d,w3d,p3d_face_w,p3d_face_s,p3d_face_l)
if not cyclic_x:
   convw,u_bc_east = modify_outlet(convw)
```

- Compute convection with new $\bar{u}$, $\bar{v}$, $\bar{w}$
- Compute continuity error

```
su3d=(convw[0:-1,:,:]-convw[1:,:,:]\ +convs[:,0:-1,:]-convs[:,1:,:]\
          +convl[:,:,0:-1]-convl[:,:,1:])/acrank/dt[itstep]
```

- $a_W, \ldots a_H$ in Poisson $\bar{p}$ eq.

```
if iter == 0 and itstep == 0:
   aw3d,ae3d,as3d,an3d,al3d,ah3d,su3d,ap3d=calcp(convw,convs,convl)
   aw3d_p=aw3d
   as3d_p=as3d
   al3d_p=al3d
```

FLOW CHART FOR $\bar{p}$/CONTINUITY EQ. IN MAIN CONT'D

- Solve Poissson $\bar{p}$ eq.

```
if solver_p == 'pyamg':
   p3d=solve_p(p3d,aw3d,ae3d,as3d,an3d,al3d, ...
elif solver_p == 'pyamgx':
# the A matrix is re-computed in solve_pyamgx every time
   aw3d,ae3d,as3d,an3d,al3d, ...
elif solver_p == 'pyamgx_p':
# the A matrix computed only once (requires more GPUY memory)
   p3d=solve_px(p3d,aw3d,ae3d,as3d,an3d,al3d,...
```

- Correct convections

```
convw,convs,convl,p3d,u3d,v3d,w3d,su3d= correct_conv(u3d,v3d,w3d ...
```

FLOW CHART FOR $k-\omega$ MODEL. IN MAIN

- Compute turbulent viscosity

```
vis3d=vist_kom(vis3d,k3d,om3d)
```

- b.c. for k

```
su3d,sp3d=bc(su3d,sp3d,k_bc_west,k_bc_east,k_bc_south,k_bc_north ...
```

- Compute $a_W \ldots a_H$ for k

```
aw3d,ae3d,as3d,an3d,al3d,ah3d,apo3d,su3d,sp3d=coeff(convw,convs,...
```

- Add generic sources, i.e. P^k and dissipation $-C_\mu k\omega$

```
su3d,sp3d,gen,comm_term=calck_kom(su3d,sp3d,k3d,om3d,vis3d ...
```

FLOW CHART FOR $k-\omega$ MODEL. IN MAIN CONT'D

- Add Crank-Nicolson

```
ap3d,su3d=crank_nicol(k3d_old,aw3d,ae3d,as3d,an3d,al3d, ...
```

- Solve k eq.

```
if solver_turb == 'pyamg':
   k3d,residual_k=solve_pyamg(k3d,aw3d,ae3d,as3d,an3d,al3d,
...
```

- The flow chart for ω eq. is the same a for k

```
def calcu(su3d,sp3d,dpdx_old,p3d_face_w,p3d_face_s):
# presssure gradient
dpdx=acrank*dphidx(p3d_face_w,p3d_face_s)+(1-acrank)*dpdx_old
su3d=su3d-dpdx*vol

# modify su & sp
su3d,sp3d=modify_u(su3d,sp3d)
# unsteady term added in crank_nicol

return su3d,sp3d
```

CALCK

```
def calck_kom(su3d,sp3d,k3d,om3d,vis3d,u3d_face_w,u3d_face_s, ...
# production term
dudx=dphidx(u3d_face_w,u3d_face_s)
dwdx=dphidx(w3d_face_w,w3d_face_s)
...

gen= (2.*(dudx**2+dvdy**2+dwdz**2)+(dudz+dwdx)**2+(dvdz+dwdy)**2+...
vist=xp.maximum(vis3d-viscos,1e-10)
su3d=su3d+vist*gen*vol
# dissipation term
if sst or kom_des:
   sp3d=sp3d-fk3d*cmu*om3d*vol
else:
   sp3d=sp3d-cmu*om3d*vol
# modify su & sp
su3d,sp3d,comm_term=modify_k(su3d,sp3d,gen)
```

PANS

- The destruction term in the ε eq. is modified: $C_{\varepsilon 2} \Rightarrow C^*_{\varepsilon 2}$. In `calceps` it reads

```
if pans:
   c2u=c_eps_1+fk3d*(fdampf2*c_eps_2-c_eps_1)
else:
   c2u=fdampf2*c_eps_2
```

- Turb. Prandtl numbers: $(\sigma_k, \sigma_\varepsilon) \Rightarrow (\sigma_k f_k^2, \sigma_\varepsilon f_k^2)$ In `coeff` it reads ($f_{k,limit}$ is, say, 0.2)

```
if prand_1 > 0:
  vis_turb=(vis3d-viscos)/prand
else:
  fk3d_local=xp.maximum(fk3d,fkmin_limit)  #this limit is used only
  vis_turb=(vis3d-viscos)/xp.abs(prand)/fk3d_local**2
```

- This array is used both PANS and DES/IDDES .... It is computed in `compute_fk`

```
def compute_fk(k3d,eps3d,fk3d):
if pans:
   L_t=k3d**1.5/eps3d
   psi=xp.maximum(1,L_t/(cdes*delta_max))
   fk3d=xp.maximum(1-(psi-1)/(c_eps_2-c_eps_1),0)
if keps_des:
   rl=k3d**1.5/eps3d
   fk3d=xp.maximum(1,rl/(0.67*delta_max))
if jl0 < 0:
   jl=xp.abs(jl0)
   fk3d[:,0:jl,:]=1
fk3d=modify_fk(fk3d)
```

- IDDES is implemented in module `modify_fk` in `hump-IDDES-ni-583-go4hybrid-mesh-STG-dt-0.002-GPU/modify_case.py`

INLET B.C.

pyCALC-LES flowchart ▸ Link

- Section 3.6
- module `modify_inlet`

pyCALC-LES flowchart ▸ Link

- Section 3.7
- module `bc`
- It is set when `u_bc_west_type`, `u_bc_east_type` ... `om_bc_high_type` are set to `'d'`

pyCALC-LES flowchart ▸ Link

- Section 5.2
- module `modify_outlet`

pyCALC-LES flowchart ▸ Link

- Section 6
- module `vist_smag`

pyCALC-LES flowchart ▸ Link

- Section 7
- module `vist_wale`

$k - \varepsilon$ Turbulence Model

pyCALC-LES flowchart ▸ Link

- Section 8
- module `calck`
- module `calceps`
- module `vist_pans`
- module `modify_eps` (in directory `channel-950-k-eps-RANS*`)

pyCALC-LES flowchart ▸ Link

- Section 10
- module `calck_kom`
- module `calcom`
- module `vist_kom`
- module `modify_om`

```
su3d[:,0,:]=ap_max*6*viscos*k3d[:,0,:]
/(c_omega_2*dist3d[:,0,:]**2)
```

WALL FUNCTIONS

▶See Section 11.14.1 in [1]

y^+: wall-adjacent cell center at $30 \leq y^+ \leq 200$

$\bar{u}$: set the wall shear stress as $\tau_w = \rho u_\tau^2$ (recall that $\rho = 1$). The log-law reads

$$\frac{\bar{u}_P}{u_\tau} = \frac{1}{\kappa} \ln\left(\frac{E u_\tau y_P}{\nu}\right)$$

where $E = 9$ and $\kappa = 0.41$.

- Can't express u_τ explicitly; we must iterate

$$u_\tau = \frac{\kappa \bar{u}_P}{\ln\left(\frac{E u_{\tau,old} y_P}{\nu}\right)}$$

k: set k at the wall-adjacent cells as $k_P = C_\mu^{-1/2} u_\tau^2$

ε: set ε at the wall-adjacent cells as $\varepsilon_P = u_\tau^3/(\kappa y_P)$

ω: set ω at the wall-adjacent cells as $\omega_P = C_\mu^{-1/2} u_\tau/(\kappa y_P)$, see Eg. 3.27 in [2]

pyCALC-LES flowchart ▸ Link

- Section 11
- module `synt_fluct` in file `synt_fluct.py`

REFERENCES

[1] L. Davidson. Fluid mechanics, turbulent flow and turbulence modeling. eBook, Division of Fluid Dynamics, Dept. of Mechanics and Maritime Sciences, Chalmers University of Technology, Gothenburg, 2021.

[2] L. Davidson. An Introduction to Turbulence Models. Technical Report 97/2, Dept. of Thermo and Fluid Dynamics, Chalmers University of Technology, Gothenburg, 1997.