Using Cantera to construct Flamelet Libraries for Large-Eddy Simulation

Carlos Pantano

Graduate Aeronautical Laboratories

CALTECH

Introduction

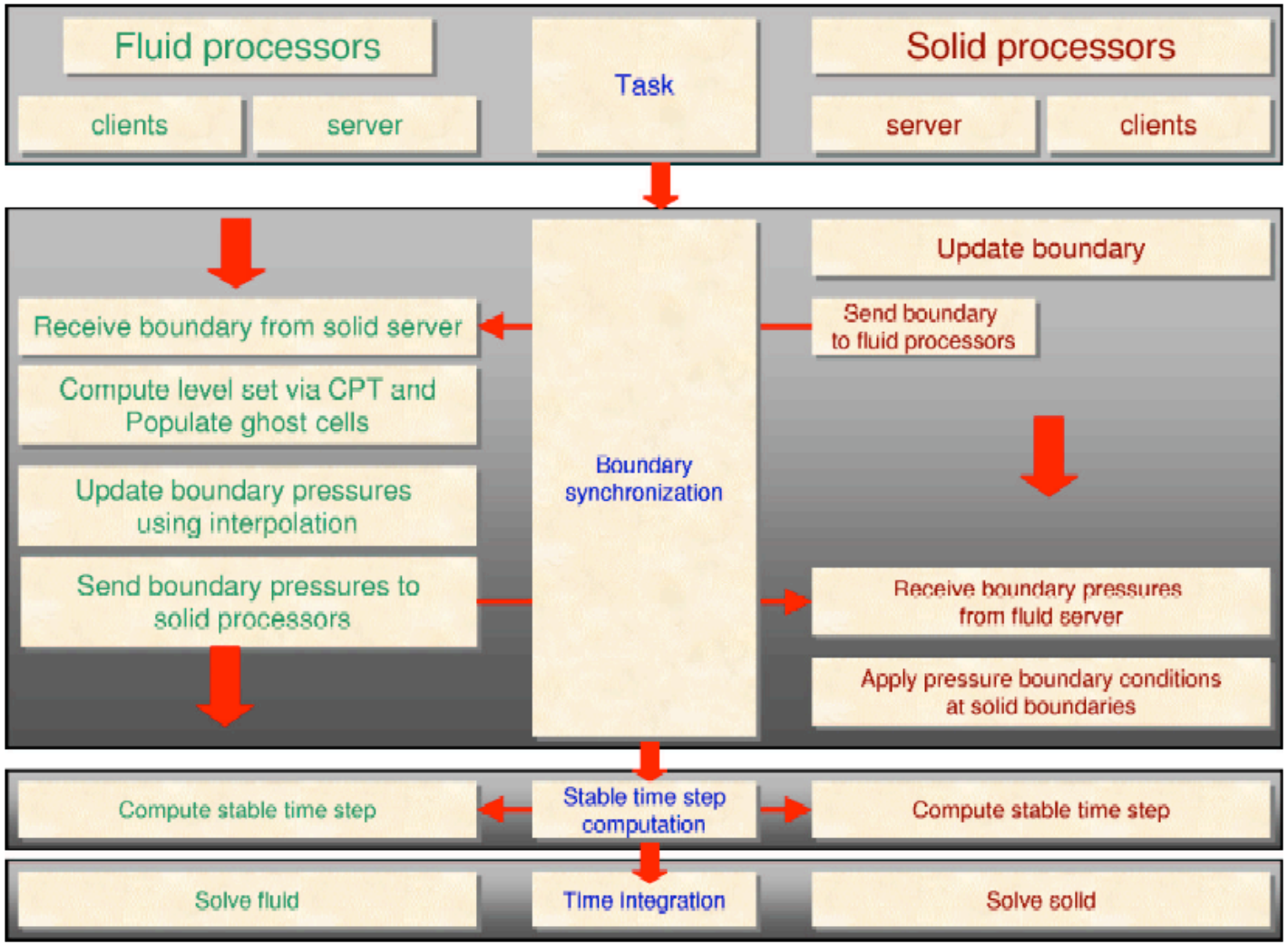
- Application of turbulent combustion physics in the development of subgrid models for LES.
- Activities:
  - Development of subgrid-scale models.
  - Numerics and implementation (patch solvers).
  - Validation of canonical flows.
- Current goal is to capture statistics of reacting shear flows (shear layers, jets and bluff body stabilized turbulent flames).

ASC™

People (Fluids Side)

- D. Meiron (ASCI)
- D.I. Pullin (SGS-LES Model), P.D. Dimotakis.
- D. Hill (Numerics), R. Deiterding (AMR).
- D. Goodwin (Cantera)

The Virtual Test Facility (VTF)



LES Formulation

- Conservation of mass, momentum and energy are expressed in the usual in terms of filtered density, $\bar{\rho}$, velocity, $\tilde{u}_i$, pressure $\bar{p}$ and the total sensible energy, with conservation equation given by

$$\frac{\partial \bar{E}}{\partial t} + \frac{\partial}{\partial x_k}\left((\bar{E} + \bar{p})\tilde{u}_k\right) = \frac{\partial}{\partial x_k}\left(\tilde{\lambda}\frac{\partial \tilde{T}}{\partial x_j}\right) + \frac{\partial}{\partial x_k}\left(\tilde{\tau}_{kj}\tilde{u}_j\right) - \frac{\partial \sigma_k^e}{\partial x_k}.$$

- The total sensible energy, $\bar{E}$ is given by

$$\bar{E} = \bar{\rho}\left(\tilde{h} + \frac{1}{2}\tilde{u}_k\tilde{u}_k\right) - \bar{p} + \bar{\rho}\tilde{k},$$

where $\tilde{k} = (\widetilde{u_i u_i} - \tilde{u}_i\tilde{u}_i)/2$ is the subgrid kinetic energy.

- The resolved shear viscosity, $\tilde{\mu}$, and thermal conductivity, $\tilde{\lambda}$, depend on temperature as $T^{0.7}$.
- The filtered enthalpy is decomposed into a resolved part and a subgrid part through,

$$\tilde{h} = \sum_{i=1}^{N} h_i(\tilde{T})\tilde{Y}_i + h_s,$$

where N is the number of species.

Closures

- The system of equations is closed by the equation of state,

$$\bar{p} = \bar{\rho} R^o \left(\tilde{T} \sum_{i=1}^{N} \frac{\tilde{Y}_i}{W_i} + w_s \right),$$

where R^o is the gas constant and W_i is the molecular weight of species i.

- The quantities that need to be modeled are

$$\begin{aligned}
\sigma_{ij} &= \bar{\rho}\left(\widetilde{u_i u_j} - \tilde{u}_i \tilde{u}_j\right), \\
\sigma_j^z &= \bar{\rho}\left(\widetilde{Z u_j} - \tilde{Z} \tilde{u}_j\right), \\
\sigma_j^e &= \bar{\rho}\left(\widetilde{h u_j} - \tilde{h} \tilde{u}_j\right) + \frac{\bar{\rho}}{2}\left(\widetilde{u_k u_k u_j} - \widetilde{u_k u_k} \tilde{u}_j\right) \\
w_s &= \sum_{i=1}^{N} \frac{\widetilde{TY}_i - \tilde{T}\tilde{Y}_i}{W_i}, \\
h_s &= \sum_{i=1}^{N} \left(\widetilde{h_i Y_i} - h_i(\tilde{T})\tilde{Y}_i\right)
\end{aligned}$$

Chemistry Model

- The governing equation of Z is

$$\frac{\partial}{\partial t}\left(\bar{\rho}\tilde{Z}\right)+\frac{\partial}{\partial x_k}\left(\bar{\rho}\tilde{Z}\tilde{u}_k\right)=\frac{\partial}{\partial x_k}\left(\bar{\rho}\tilde{D}\frac{\partial\tilde{Z}}{\partial x_i}\right)-\frac{\partial\sigma_k^z}{\partial x_k},$$

- Combustion is modeled through the conserved scalar approach with an assumed Beta-pdf is used to close the problem

$$\tilde{P}_{sgs}(Z;x,t)=\frac{Z^{n_1-1}(1-Z)^{n_2-1}}{B(n_1,n_2)}.$$

- Averages are obtained from

$$\tilde{Y}_i(x,t)=\int_0^1 Y_i^f(Z)\tilde{P}_{sgs}(Z;x,t)dZ.$$

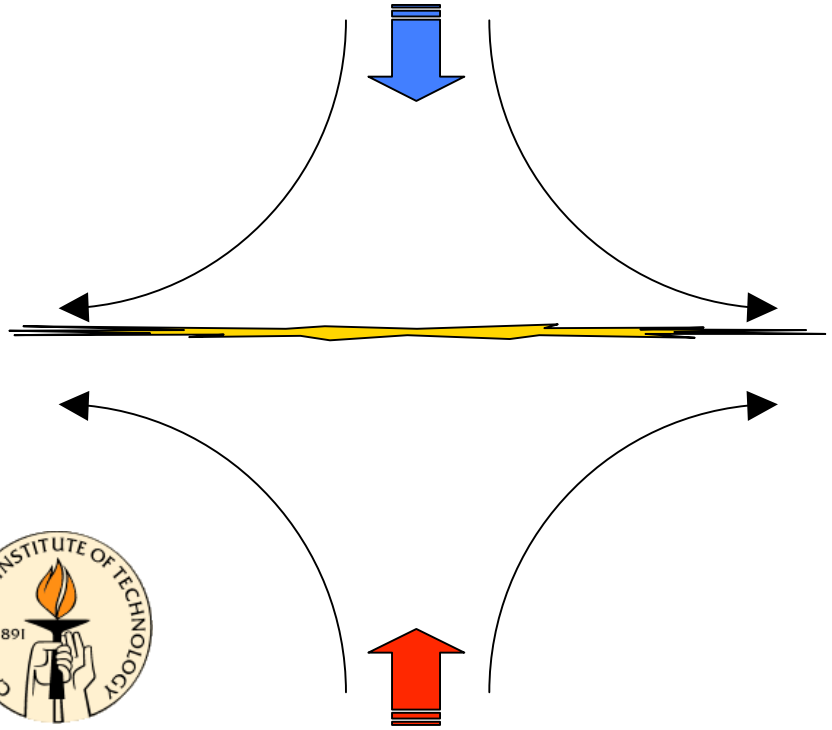
- This can be used to obtain h_s and w_s,

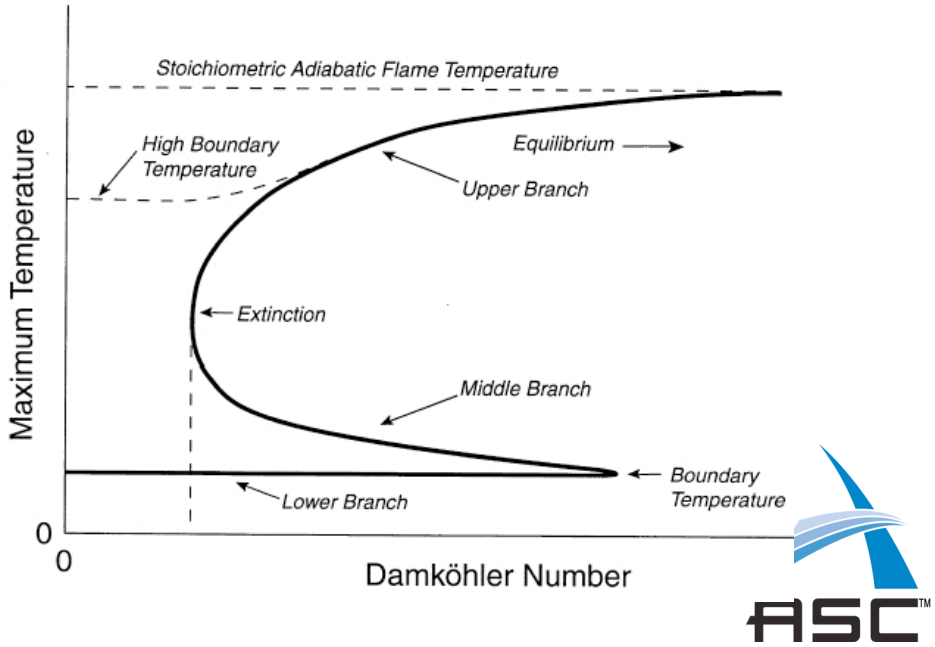
$$w_s(x,t)=\sum_{i=1}^{N}\int_0^1\frac{T^f(Z)Y_i^f(Z)-\tilde{T}\tilde{Y}_i}{W_i}\tilde{P}_{sgs}(Z;x,t)dZ.$$

Flamelet Libraries with Cantera

- 1D counterflow diffusion flame problem is used to construct a flamelet library that is then loaded in the LES solver.
- The single counterflow flame example can be found in:

Cantera/python/examples/npflame1.py





Actual Code Example

Reference Conditions

```
# Initialization of Reference conditions
p          =   OneAtm          # pressure
tin_f      =   300.0           # fuel inlet temperature
tin_o      =   300.0           # oxidizer inlet temperature
mdot_o     =   0.5             # kg/m^2/s
mdot_f     =   0.25            # kg/m^2/s
L          =   0.02            # m

comp_o      =  'O2:0.21, N2:0.78, AR:0.01';   # air composition
comp_f      =  'CH4:1';                       # fuel composition

# distance between inlets is 2 cm; start with an evenly-spaced 11-point grid
initial_grid = L*array([0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0],'d')
Nfl=80              # number of flamalets to compute
```

Actual Code Example

Load Mechanism

```
# Here we use GRI-Mech 3.0 with mixture-averaged transport properties.
gas = GRI30('Mix')

# Number of species
nsp = gas.nSpecies()

# Number of elements
nel = gas.nElements()

# Molar Masses
mmas = gas.molarMasses()

# Atomic Weights
awgt = gas.atomicWeights()
```

Actual Code Example

Auxiliary Functions

```
# Define auxiliary function to write flamelet solutions into your library file.

def writeFlamelet(f, list):
    for item in list:
        if type(item) == types.StringType:
            f.write(item+' ')
        else:
            f.write(`item`+' ')
    f.write('\n')

# Open flamelet library file and write number of solutions inside
flt = open('ch4.flt','w')
flt.write('N = ' + str(Nfl) + '\n')
```

Actual Code Example

Initialize Flame

```
# Create flame
f = CounterFlame(gas = gas, grid = initial_grid)
# set the error tolerances
f.set(tol = tol_ss, tol_time = tol_ts)
f.fuel_inlet.set(massflux = mdot_f,
                 mole_fractions = comp_f,
                 temperature = tin_f)

f.oxidizer_inlet.set(massflux = mdot_o,
                     mole_fractions = comp_o,
                     temperature = tin_o)

# construct the initial solution estimate. To do so, it is necessary to specify the fuel
# species.
f.init(fuel = 'CH4')
```

Actual Code Example

```
# Set the state of the two inlets
ampl = 1.0
for ix in range(Nfl):
    mflux = mdot_f*ampl
    f.fuel_inlet.setMdot(mflux)
    mflux = mdot_o*ampl
    f.oxidizer_inlet.setMdot(mflux)

   # First disable the energy equation and solve the problem without refining the grid
    f.set(energy = 'off')
    f.solve(loglevel, 0)

   # Now specify grid refinement, turn on the energy equation, and solve again.
    f.setRefineCriteria(ratio = 200.0, slope = 0.1, curve = 0.2, prune = 0.02)
    f.set(energy = 'on')
    f.solve(1)
```

Actual Code Example

```
# write solution
 nz = f.flame.nPoints()
 z = f.flame.grid()
 T = f.T()
 u = f.u()
 V = f.V()

 for n in range(nz):
     f.setGasState(n)
     Y = gas.massFractions()
     # Define Bilger mixture fraction
    Zbilger = BilgerMixtureFraction(Y)

    writeFlamelet(flt, [Zbilger, z[n], u[n], V[n], T[n]]+list(gas.massFractions()))
```

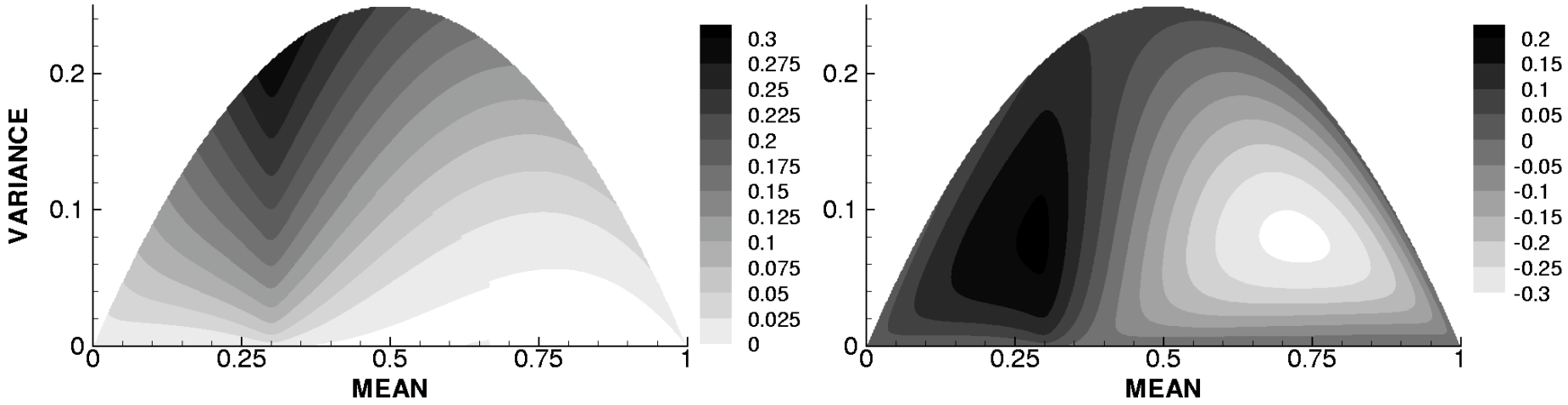
Actual Code Example

```
   # Increment mass flux amplification factor
   ampl = ampl + 0.2

# End of loop and close of flamalet library file
flt.close()
```

- Once the elemental flamelets are computed, a library with entry variables of mean mixture fraction and variance can be constructed by performing the appropriate weighting with the assumed PDF.

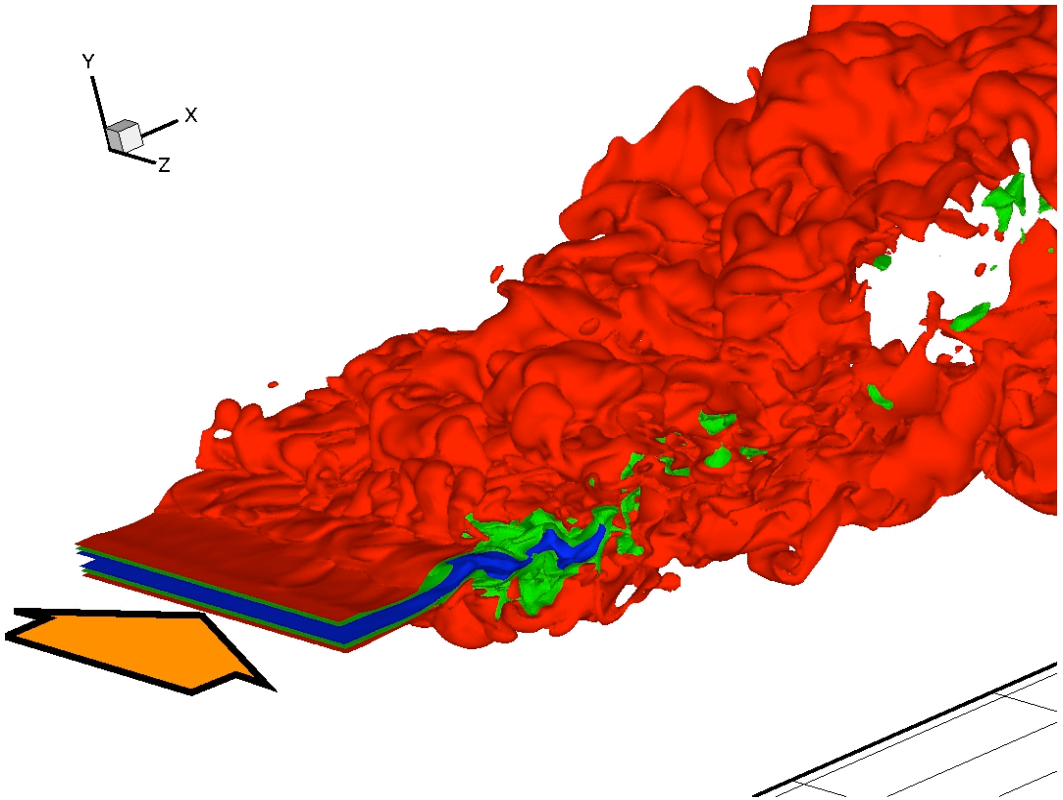
Model Subgrid terms



h_s w_s

These pre-computed tables are then loaded in the LES solver.

Turbulent reacting jet



- Fully compressible reactive LES solver with arbitrary complex chemistry using flamelet libraries (Cantera).

Conclusions

- Constructing flamelet libraries using the Python interface to Cantera is very convenient.
- No need to compile code, everything is accomplished through scripting.
- Python lets you make calls to other libraries directly so it can be easily integrated with other numerical subroutines.
- Flamelet libraries of any complex chemistry can be generated with little effort.