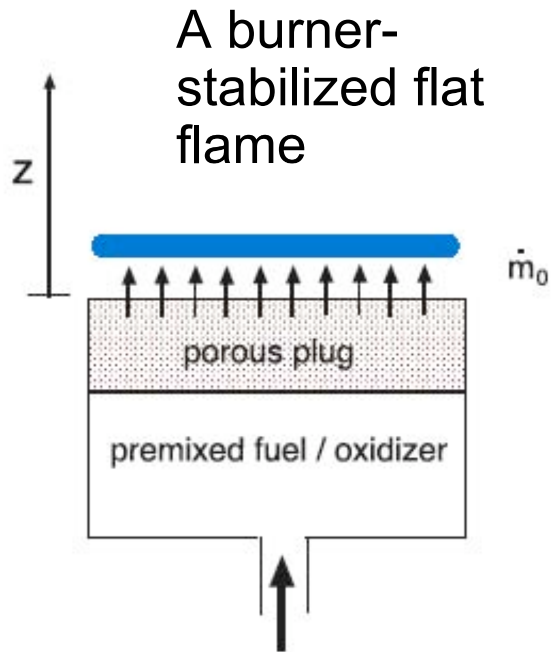
One-Dimensional Flames

David G. Goodwin

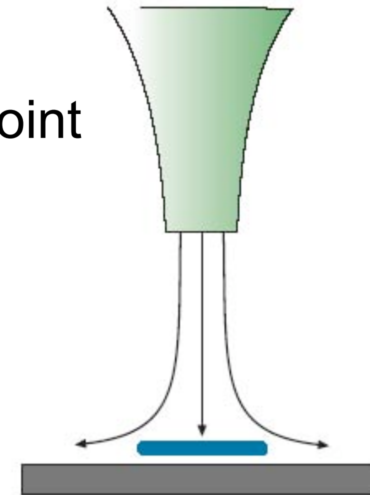
Division of Engineering and
Applied Science
California Institute of Technology

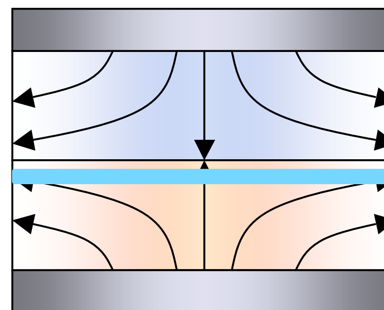
Several types of flames can be modeled as "one-dimensional"

A burner-stabilized flat flame



A premixed stagnation-point flame



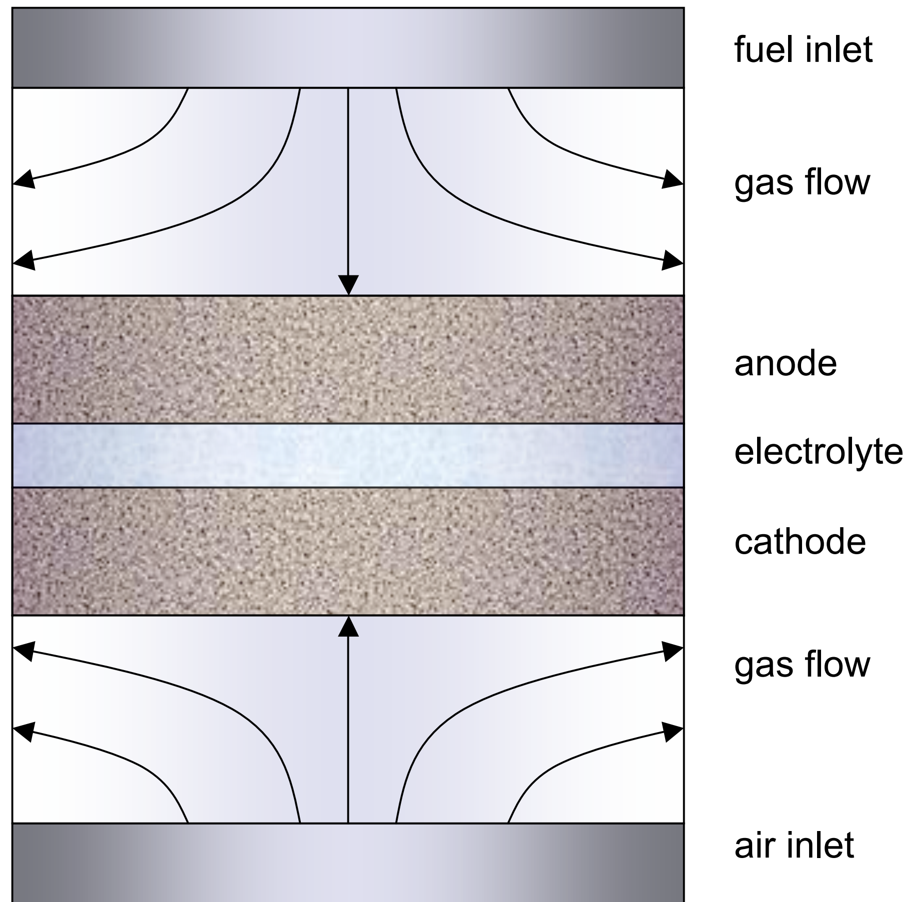


A non-premixed counterflow flame

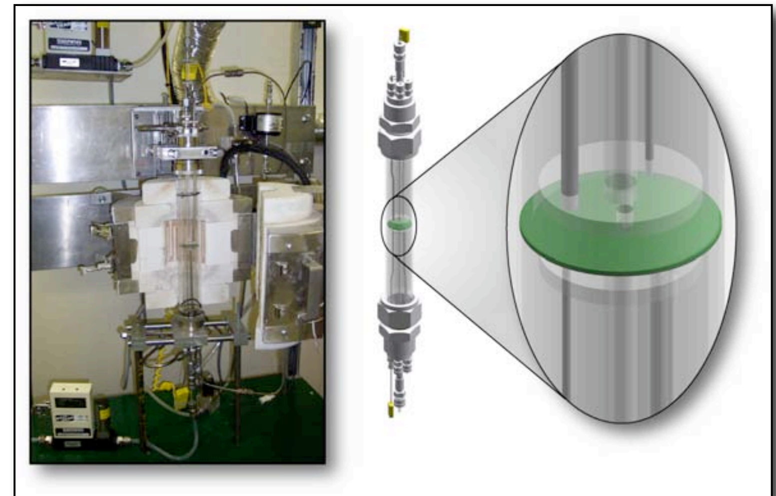
One-Dimensionality

- These flames are 1D in the sense that, when certain conditions are fulfilled, the governing equations reduce to a system of ODEs in the axial coordinate
- This occurs either because the flow is physically 1D (no radial velocity component), or...
- The flow is physically 2D, but a similarity transformation reduces the problem dimensionality to 1D

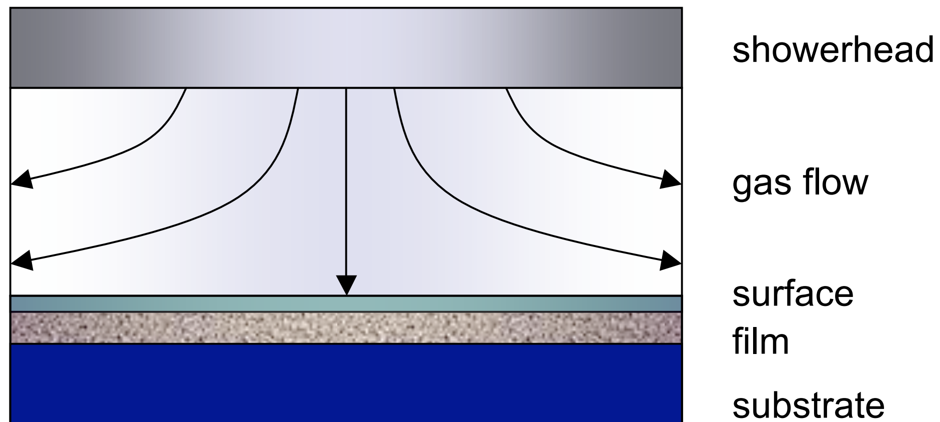
One-dimensional flames are only one type of 1D reacting-flow problem



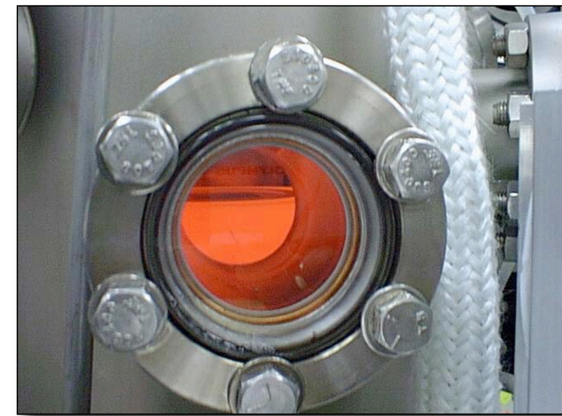
Fuel cell test facility

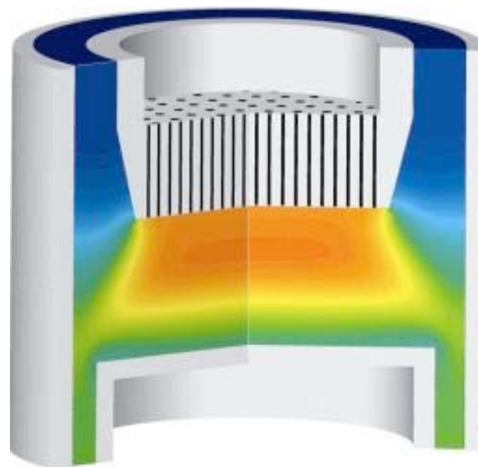


Thin film deposition

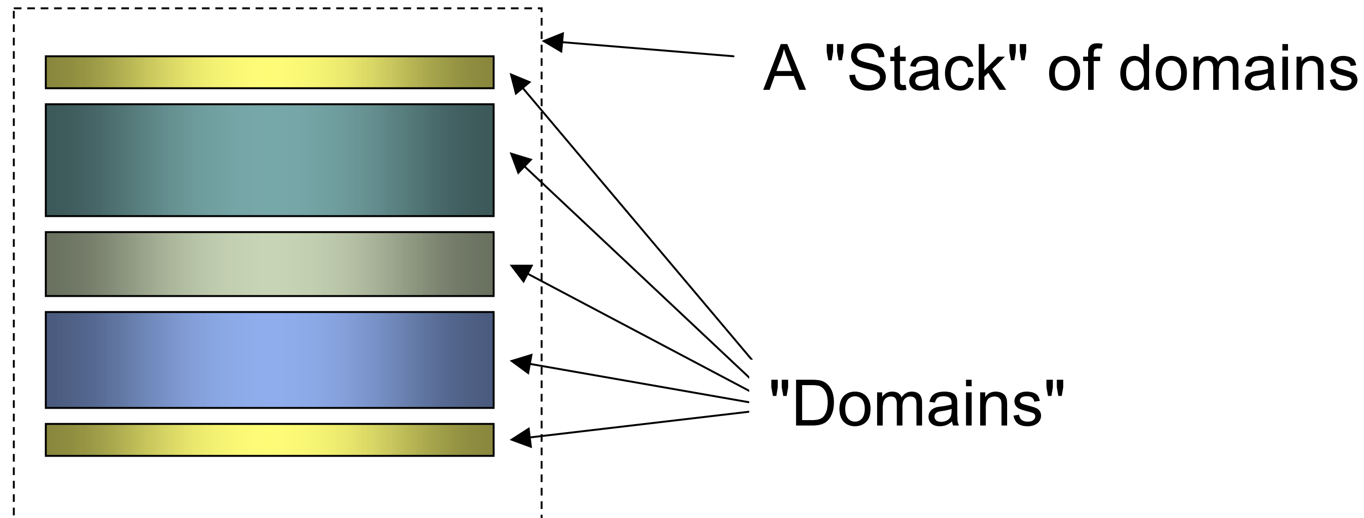


Stagnation-flow chemical vapor deposition reactor





Cantera provides capabilities to solve general axisymmetric 1D problems



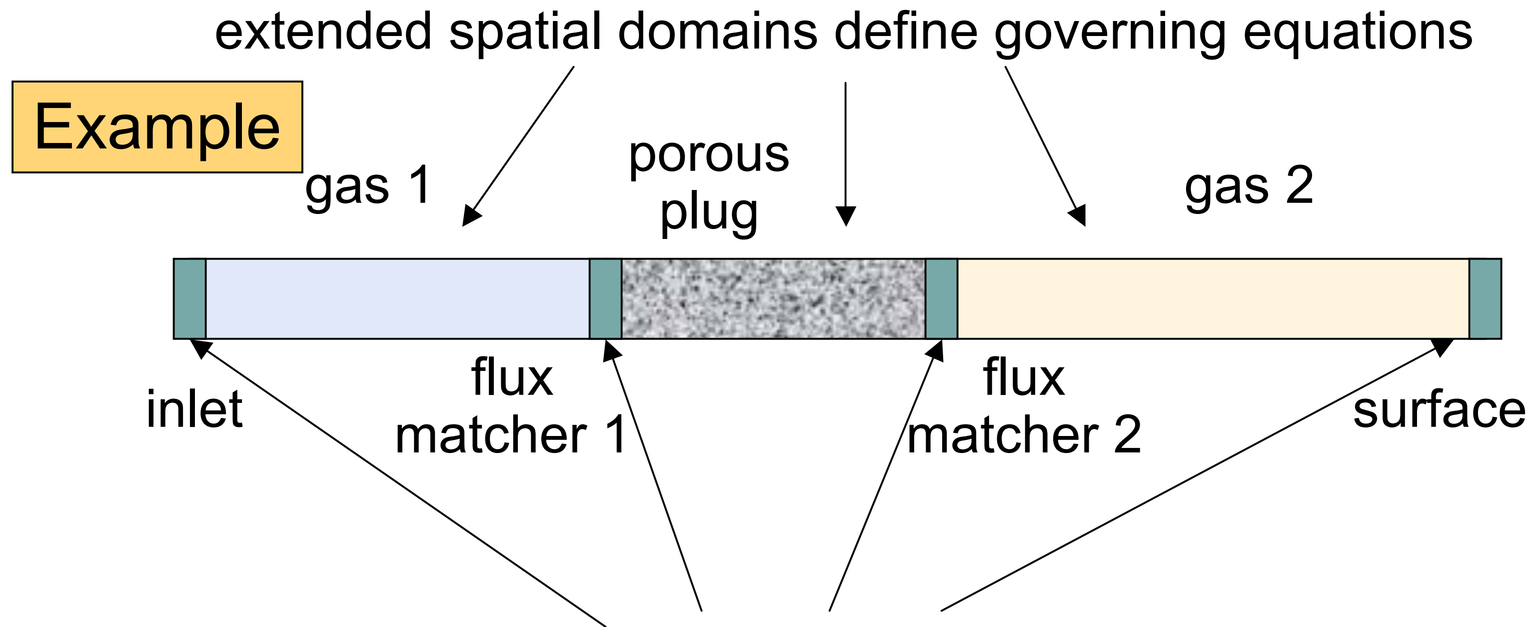
- Each domain represents a distinct phase, flowfield, or interface
  - a gas flow
  - an inlet or outlet
  - a surface
  - a solid ...

Multi-Physics Simulations

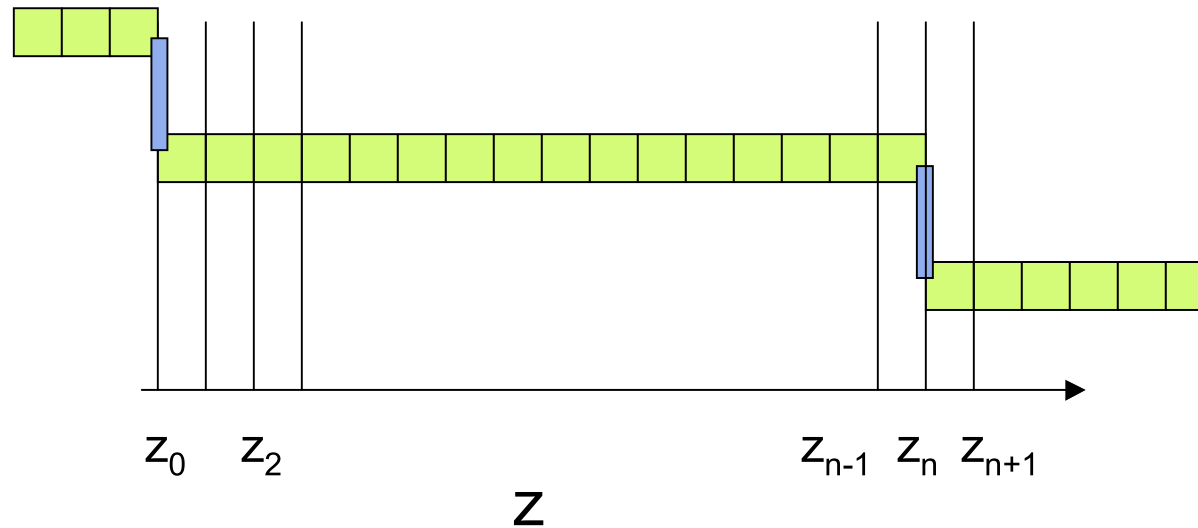
- Physics may be different in each domain
- Each domain has its own set of variables (components) and governing equations
- Spatially-extended domains alternate with "connector" or "boundary" domains that provide the coupling
- Solution determined for all domains simultaneously in fully-coupled, fully-implicit way

General Structure

- Applies for any 1D problem: flame, fuel cell stack, CVD stagnation flow, ...
- A 1D problem is partitioned into *domains*



Grids imposed on spatial domains



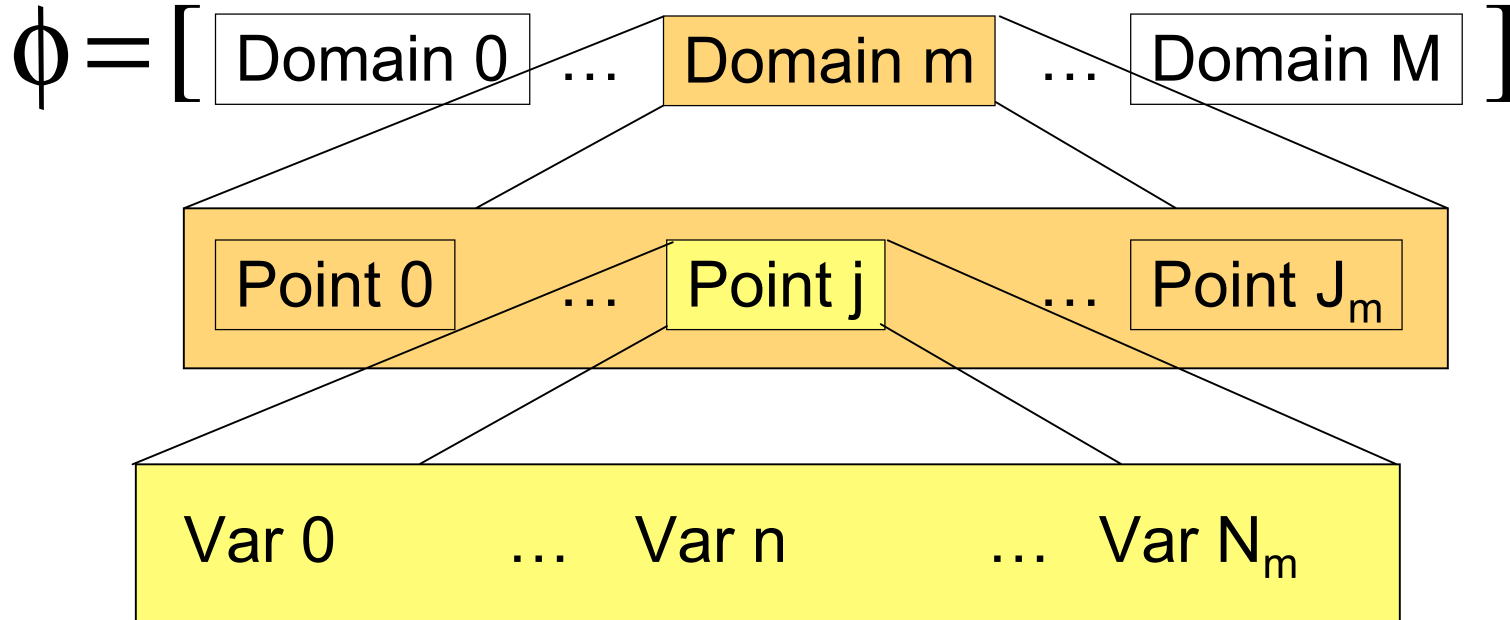
spatial domain boundaries coincide

each domain refines its grid separately to resolve its solution

Domain Variables

- Each type of domain has a specified set of variables at each grid point
- Examples
  - an axisymmetric flow domain -
    - K + 4 variables per point
    - u, V, T, Λ, Y1, ..., YK
  - a surface domain
    - K + 1 variables at one point only
    - T, coverages of all surface species

Solution Vector



Variables are ordered first by domain, then by point in the domain. Note that each domain *m* may have a different number of points J_m, and a different number of variables per point N_m. All ordering is done left-to-right.

Solution Method

- Finite-difference flow equations to form a system of nonlinear algebraic equations
- Use a hybrid Newton / time-stepping algorithm to solve the equations
- Adaptively refine / coarsen grid to resolve the profiles, or remove unnecessary points if over-resolved

Residual Function

- In each domain, there is an equal number of equations and unknowns at each point

- The nth equation at the jth point in the mth domain has the form

$$F_{j,m,n}(\phi) = 0$$

  where $F_{j,m,n}$ depends only on solution variables at points j, j-1, and j+1

- Therefore, the Jacobian of this system of equations is banded.

The residual equations are solved using a variant of Newton's Method

Classical Newton's method:

linearize about solution estimate $\phi^{(0)}$:

$$F_{lin,i}^{(0)} = F_i(\phi^{(0)}) + \sum_{\mathrm{j}} \left.\frac{\partial F_i}{\partial \phi_j}\right|_{\phi=\phi^{(0)}} (\phi_j - \phi_j^{(0)})$$

solve linear problem to generate new estimate of ϕ:

$$\mathbf{F}_{lin}(\phi^{(1)}) = \mathbf{0},$$

$$\phi^{(1)} = \phi^{(0)} - \left[\mathbf{J}^{(0)}\right]^{-1} \mathbf{F}^{(0)}$$

where $J_{i,j} = \partial F_i / \partial \phi_j$

Quadratic convergence

- If F is linear, this leads to the exact solution ϕ^* in 1 step
- If F is *quadratic*, then repeating this process produces a convergent sequence of solution estimates $\phi^{(0)}, \phi^{(1)}, \phi^{(2)}, \phi^{(3)}, \ldots$

  with the error decreasing *quadratically*:

$$\lim_{n \to \infty} \left| \phi^{(n+1)} - \phi^* \right| = A \left| \phi^{(n)} - \phi^* \right|^2$$

Transient Problem

- If Newton iteration fails to find the steady-state solution, we attempt to solve a psuedo-transient problem with a larger (perhaps much large) domain of convergence
- This problem is constructed by adding transient terms in each conservation equation where this is physically reasonable
- This may not be possible for algebraic constraint equations; these are left unmodified

$$A \frac{d\underset{\sim}{\phi}}{dt} = F(\underset{\sim}{\phi})$$

$$F(\phi^{(n+1)}) - A\frac{\phi^{(n+1)} - \phi^{(n)}}{\Delta t} = 0$$

- Let A be a diagonal matrix with 1 on the diagonal for those equations with a transient term, and 0 on the diagonal for constraint equations.
- Then the modified problem is as shown above.

Transient Problem (cont'd)

- This is of the form

$$F_{transient}(\phi^{(n+1)};\phi^{(n)},\Delta t)=0$$

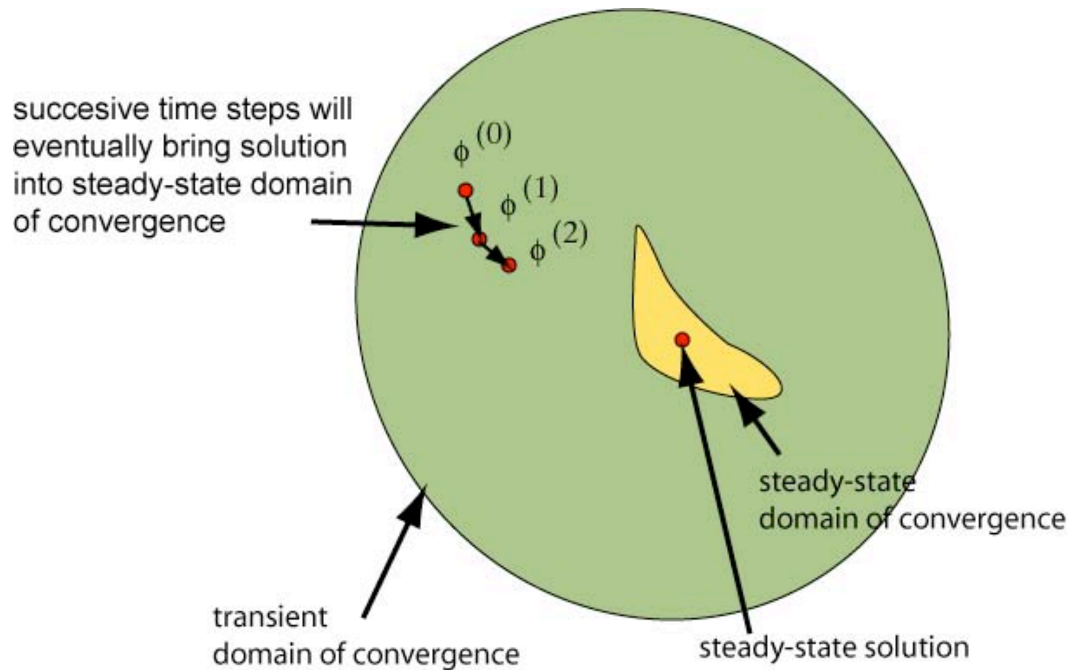
- Note that if A = I (all equations have transient terms), then for sufficiently small time step size this transient residual function approaches a linear problem

- In this case, there will be some non-zero Δt for which the Newton algorithm converges for the transient problem.

- But note: if A has zeros on diagonal (algebraic constraints), and the initial solution does not satisfy these constraints, then there is no guarantee that the transient Newton problem will converge, no matter how small the step size

- In this case, there is nothing to do but try to generate by some other means a better starting estimate that more nearly satisfies the algebraic constraints

Time step until solution enters s.s. domain of convergence, then proceed to solution using s.s. Newton



- Take a few time steps
- Try to solve steady-state problem
- If not yet in steady-state domain of convergence, take a few more time steps
- Repeat until steady-state Newton succeeds

A larger domain of convergence is achieved by using a damped Newton method

- Compute a Newton step
- If the step carries the solution outside prescribed limits, determine the scalar multiplier required to bring it back in
- Starting with this (possibly scaled) new solution vector, backtrack along the Newton direction until a point is found where the next Newton step would have a smaller norm than the original undamped, unscaled Newton step
- If such a point can be found, accept the damped Newton step
- Otherwise abort and try time-stepping for a while
- Repeat until the solution converges, or a damped Newton step fails.

The Jacobian

- By far the most CPU-intensive operation in this algorithm is evaluating the Jacobian matrix
- Exact Jacobians are not required, so try to re-use previously-computed Jacobians
- Only recompute J if:
  - The damped Newton algorithm failed, and the Jacobian is out-of-date, or
  - a specified maximum number of times it may be used has been reached
- Note that switching between transient and steady-state modes only adds/subtracts a constant from the diagonal; no need to recompute Jacobian just to go from steady to transient or vise versa.

Algorithm comparison to TWOPNT

- The numerical method used is similar to hybrid Newton/time-stepping schemes used by others. In particular, it draws on the report by Grcar (Sandia Report SAND91-8230, 1992)

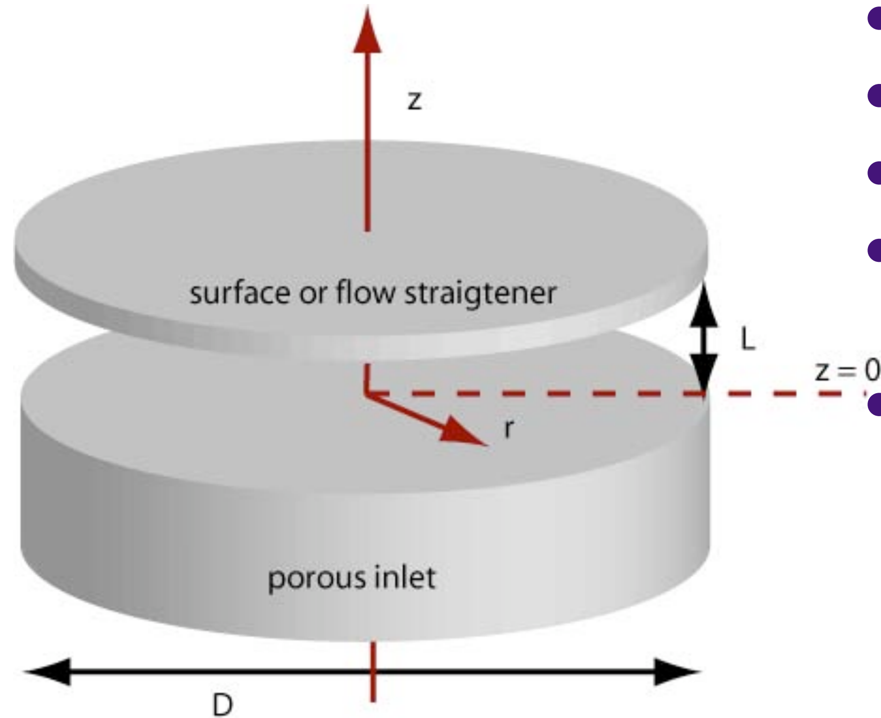
- Differences:
  - Works for arbitrary multi-domain problems
  - Jacobian not recomputed when switching between transient and steady-state modes (only diagonal terms modified)
  - RMS weighted error norm used, rather than max value norm
  - Grid points can be automatically removed as well as added
  - Greater control over number of time steps between steady-state Newton solution attempts

The AxisymmetricFlow Domain Type

Python / MATLAB: class AxisymmetricFlow

C++: class AxiStagnFlow

Axisymmetric flow geometry and variables



- u = axial velocity
- v = radial velocity
- T = temperature
- Y_k = mass fraction of species k

- Boundary conditions at z = 0 and z = L:
  - u, T, Y_k independent of r
  - v linear in r (usually zero)

- Low Mach number: P nearly constant

Similarity Solution

- Consider the limits
  - $L/D \ll 1$
  - $Ma \ll 1$

- If these limits are satisfied, and if the boundary conditions are satisfied, then the *exact* flow equations admit a solution with the properties:
  - $u = u(z)$
  - $v = rV(z)$
  - $T = T(z)$
  - $Y_k = Y_k(z)$
  - $P = P_0 + \Lambda r^2/2$.

- Here Λ is a constant that must be determined as part of the solution.

For conditions where similarity solution holds, flow equations reduce to ODEs in axial coordinate z

Continuity

$$\frac{d}{dz}(\rho u) + 2\rho V = 0$$

Radial Momentum

$$\rho\frac{dV}{dt} = \frac{d}{dz}\left(\mu\frac{dV}{dz}\right) - \Lambda - \rho u\frac{dV}{dz} - \rho V^2$$

Species

$$\rho\frac{dY_k}{dt} = -\rho u\frac{dY_k}{dz} - \frac{dj_k}{dz} + W_k\dot{\omega}_k$$

Energy

$$\rho c_p\frac{dT}{dt} = -\rho c_p u\frac{dT}{dz} + \frac{d}{dz}\left(\lambda\frac{dT}{dz}\right) - \sum_k W_k\dot{\omega}_k h_k - \sum_k j_k c_{p,k}\frac{dT}{dz}$$

Upwind differencing for convective terms

Continuity

$$\frac{d}{dz}(\rho u) + 2\rho V = 0$$

Radial Momentum

$$\rho\frac{dV}{dt} = \frac{d}{dz}\left(\mu\frac{dV}{dz}\right) - \Lambda - \rho u\boxed{\frac{dV}{dz}} - \rho V^2$$

Species

$$\rho\frac{dY_k}{dt} = -\rho u\boxed{\frac{dY_k}{dz}} - \frac{dj_k}{dz} + W_k\dot{\omega}_k$$

Energy

$$\rho c_p\frac{dT}{dt} = -\rho c_p u\boxed{\frac{dT}{dz}} + \frac{d}{dz}\left(\lambda\frac{dT}{dz}\right) - \sum_k W_k\dot{\omega}_k h_k - \sum_k j_k c_{p,k}\boxed{\frac{dT}{dz}}$$

if $u_j > 0$:

$$\left(\frac{df}{dz}\right)_j = \frac{f_j - f_{j-1}}{z_j - z_{j-1}}$$

otherwise:

$$\left(\frac{df}{dz}\right)_j = \frac{f_{j+1} - f_j}{z_{j+1} - z_j}$$

Central differencing for diffusive terms

Continuity

$$\frac{d}{dz}(\rho u) + 2\rho V = 0$$

Radial Momentum

$$\rho \frac{dV}{dt} = \boxed{\frac{d}{dz}\left(\mu \frac{dV}{dz}\right)} - \Lambda - \rho u \frac{dV}{dz} - \rho V^2$$

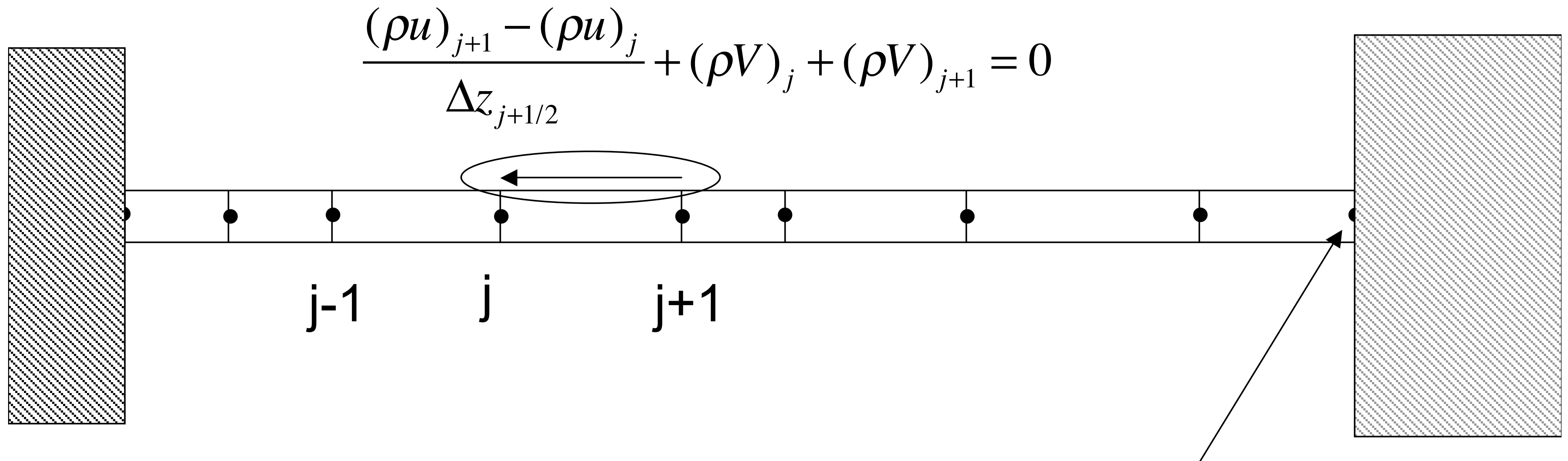
Species

$$\rho \frac{dY_k}{dt} = -\rho u \frac{dY_k}{dz} - \boxed{\frac{dj_k}{dz}} + W_k \dot{\omega}_k$$

Energy

$$\rho c_p \frac{dT}{dt} = -\rho c_p u \frac{dT}{dz} + \boxed{\frac{d}{dz}\left(\lambda \frac{dT}{dz}\right)} - \sum_k W_k \dot{\omega}_k h_k - \sum_k j_k c_{p,k} \frac{dT}{dz}$$

Axial velocity information flow

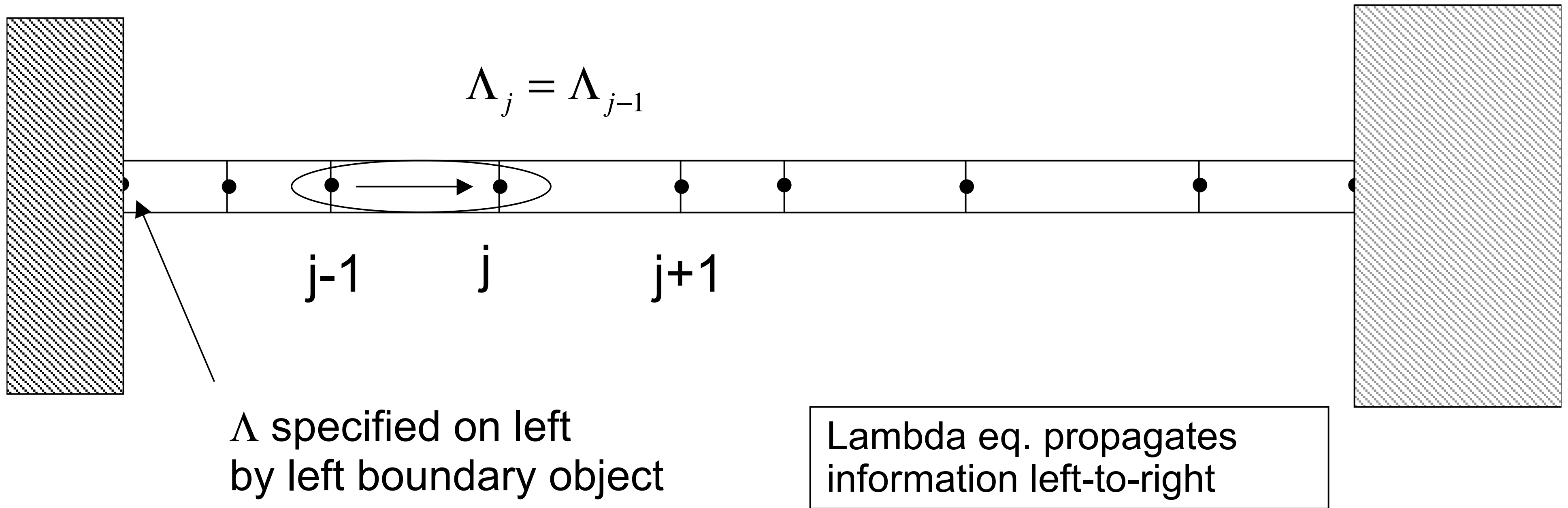


Continuity eq. propagates information right-to-left

ρu specified on right by right boundary object

burner-stabilized flame: zero gradient
counterflow flame: specified value
stagnation-point flame: zero

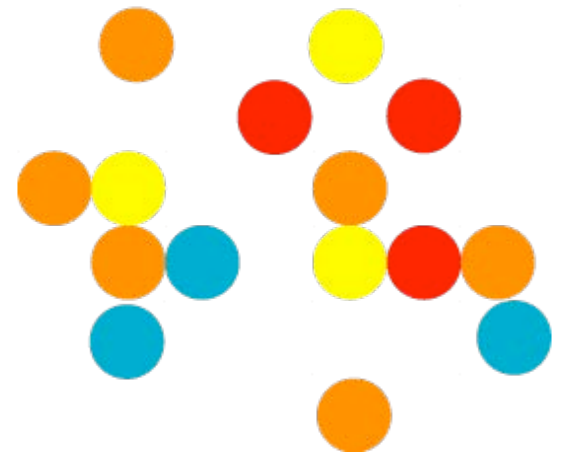
Lambda Equation



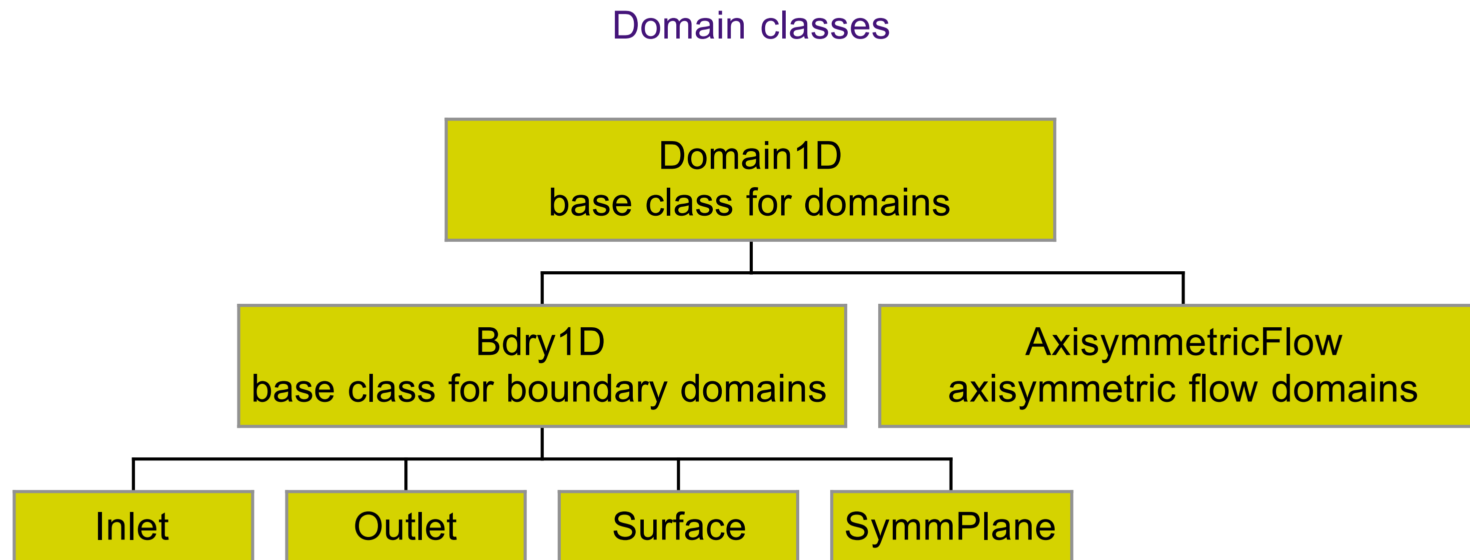
If mass flow rate from left is specified, then residual equation for Λ at left is

$$(\rho u)_0 = \dot{m}_{left}$$

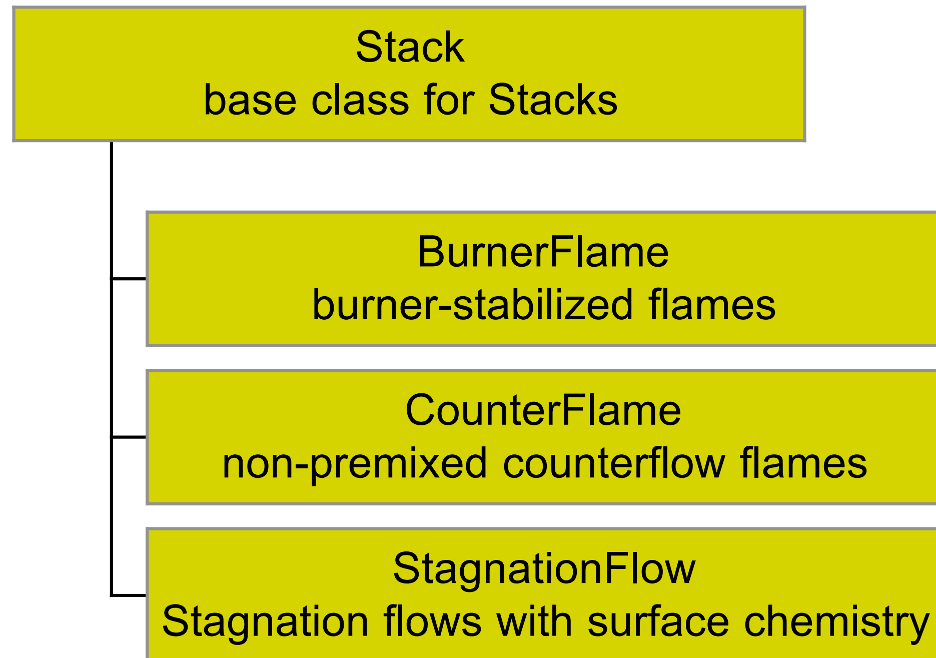
Flame Simulations in Python



Domain Class Hierarchy



Stack Class Hierarchy



Boundary Class Properties

- Inlet:
  - specified T, V, Y_k
  - mass flux specified via Λ (left inlet) or directly (right inlet)g
- Outlet
  - zero for V and Λ
  - zero gradient for u, T, Y_k
- Symm1D
  - zero u,
  - zero gradient for everything else

Surface Boundary Class

- Surface species coverages

$$\dot{s}_j = 0$$

- Coupling to gas
  - Specified T, u = 0*, V = 0
  - Species:

$$j_k + \dot{s}_k W_k = 0$$

  - *to be modified to handle the case of net mass deposition or etching

Flame Simulations in Python: a Burner-Stabilized Flame

flame1.py

```
#
# FLAME1 - A burner-stabilized flat flame
#
#    This script simulates a burner-stablized lean
#    hydrogen-oxygen flame at low pressure.
#
from Cantera import *
from Cantera.OneD import *

################################################################
#
# parameter values
#
p          =   0.05*OneAtm          # pressure
tburner    =   373.0                # burner temperature
mdot       =   0.06                 # kg/m^2/s

rxnmech    =  'h2o2.cti'            # reaction mechanism file
mix        =  'ohmech'              # gas mixture model
comp       =  'H2:1.8, O2:1, AR:7'  # premixed gas composition
```

flame1.py

```
# The solution domain is chosen to be 50 cm, and a point very near the
# downstream boundary is added to help with the zero-gradient boundary
# condition at this boundary.
initial_grid = [0.0, 0.02, 0.04, 0.06, 0.08, 0.1,
                0.15, 0.2, 0.4, 0.49, 0.5]  # m

tol_ss    = [1.0e-5, 1.0e-13]       # [rtol atol] for steady-state
                                    # problem
tol_ts    = [1.0e-4, 1.0e-9]        # [rtol atol] for time stepping

loglevel  = 1                       # amount of diagnostic output (0
                                    # to 5)

refine_grid = 1                     # 1 to enable refinement, 0 to
                                    # disable
```

flame1.py

```
################ create the gas object ########################
#
# This object will be used to evaluate all thermodynamic, kinetic,
# and transport properties
#
gas = IdealGasMix(rxnmech, mix)

# set its state to that of the unburned gas at the burner
gas.set(T = tburner, P = p, X = comp)

f = BurnerFlame(gas = gas, grid = initial_grid)

# set the properties at the burner
f.burner.set(massflux = mdot, mole_fractions = comp,
             temperature = tburner)
```

flame1.py

```
f.set(tol = tol_ss, tol_time = tol_ts)
f.setMaxJacAge(5, 10)
f.set(energy = 'off')
f.init()
f.showSolution()

f.solve(loglevel, refine_grid)

f.setRefineCriteria(ratio = 200.0, slope = 0.05, curve = 0.1)
f.set(energy = 'on')
f.solve(loglevel,refine_grid)

f.save('flame1.xml')
f.showSolution()
```

flame1.py

```
 write the velocity, temperature, and mole fractions to a CSV file
z = f.flame.grid()
T = f.T()
u = f.u()
V = f.V()
fcsv = open('flame1.csv','w')
writeCSV(fcsv, ['z (m)', 'u (m/s)', 'V (1/s)', 'T (K)', 'rho (kg/m3)']
         + list(gas.speciesNames()))
for n in range(f.flame.nPoints()):
    f.setGasState(n)
    writeCSV(fcsv, [z[n], u[n], V[n], T[n], gas.density()]
             +list(gas.moleFractions()))
fcsv.close()

print 'solution saved to flame1.csv'

f.showStats()
```

Where to find files in the source distribution

- C++
  - Directory Cantera/src/oneD
- Python
  - Modules in directory Cantera/python/Cantera/OneD
  - Module onedim.py in this directory contains domain and stack classes
  - Extension module source file ctonedim_methods.cpp in directory Cantera/python/src
- MATLAB
  - m-files in in directory Cantera/matlab/cantera/1D
  - class Domain1D in Cantera/matlab/cantera/1D/@Domain1D
  - class Stack in Cantera/matlab/cantera/1D/@Stack
  - MEX source file Cantera/matlab/cantera/private/onedimmethods.cpp
- clib
  - File Cantera/clib/src/ctonedim.cpp