PERFORM
Prototyping Environment for Reacting Flow Order Reduction Methods

User Guide

Christopher R. Wentland,* Ashish S. Nair, Elnaz Rezaian,
Cheng Huang, Karthik Duraisamy

February 3, 2021

*Email: chriswen [at] umich [dot] edu

Contents

1 Introduction

This document serves as a reference for users in setting up and running 1D reacting compressible flow simulations using `PERFORM`. Details on running the code and all necessary input files are provided. All input parameters, acceptable parameter formats, and parameter defaults (where applicable) are described. If you would like details on solver methods implemented in `PERFORM`, please refer to the general theory documentation.

Instructions for installing and executing `PERFORM` are provided in Section 2. In order to run full-order model simulations, four input files are required: a `solverParams.inp` file (Section 3), a gas file (Section 4), a mesh file (Section 5), and an initial conditions file (Section 6). In order to run reduced-order model simulations, an additional `romParams.inp` file (Section 7) is required.

Details on the pre-built examples (found in the `perform/examples` directory) can be found in Section 9, along with sample plots which should be automatically generated by running the examples with no changes to the input files (beyond specifying the locations of the gas file, mesh file, and initial condition files in `solverParams.inp`).

1.1 Bugs, Issues, and Contributing

If you experience errors or unexpected solver behavior when running `PERFORM`, please first double-check your input parameters and use this document as a reference for proper input file formatting. If problems persist, please create a new issue on the code repository, and I'll do my best to resolve it. However, if a submitted issue is related to a significant *expansion* of code capabilities (e.g. adding a new ROM model or flux scheme), I probably won't work on it.

On the other hand, if you would like to personally work to expand the code and contribute to `PERFORM`, first of all thank you! Please email me to request permission to contribute. After that, feel free to create a new branch and submit a pull request when you feel the new additions are complete. I have yet to implement any automatic testing for the repository, so you would be doing me a major solid by checking that your changes are compatible with **all** of the example cases before creating a pull request. Under no circumstance should you try to push any binary or image data without explicit permission, as this just makes the Git history unnecessarily large. I'll just tell you that you need to make a new branch without those files, delete your branch from the Github repository, and run garbage collection.

1.2 General Tips and Notes

Below is a list of general tips for running `PERFORM` and some notes on the solver. This list will be periodically updated as I think of more.

1. Native plotting of field/probe data via `matplotlib` during runtime is typically the single most time-intensive operation. As such, I would recommend setting `visInterval` in `solverParams.inp` to the largest plotting interval that you are comfortable with while setting up a new case, toying with solver parameters, and visually monitoring the unsteady probe and field plots. When you have finalized the case and want to write images to disk (`visSave = True`) for your own results reporting, you should then set `visInterval` to whatever you want. The current default value of `visInterval = 1` makes the code atrociously slow, so please make sure to set this value manually.

2. The current default residual tolerance for implicit time integrators of 1e-12 is probably a little too strict for some problems. You might get away with manually setting a smaller residual tolerance for quicker convergence.

3. When initializing a case with a step function (e.g., using `icParamsFile` in `solverParams.inp`, as in the case of the Sod shock tube example) with a second-order flux scheme, make sure you're applying a gradient limiter (e.g., `gradLimiter = "barth"`), or the solution is likely to explode immediately.

4. I'm aware of the non-physical pressure bump produced at a premixed flame. This is a function of a bulk density mismatch with species densities resulting from our approximation of the diffusion velocity term in the viscous flux. We have a correction term which helps mitigate this, but it's not perfect. Ultimately, the pressure bump is usually $\sim$1e-5% of the mean pressure. This code isn't intended for solving things like counterflow diffusion flames with very high accuracy; you should use FlameMaster for that.

2 Installing and Running

To clone the PERFORM source code repository, execute the following command from your terminal command line

```
git clone https://github.com/cwentland0/perform.git
```

As of the writing of this section, PERFORM is currently installed locally via pip (or pip3, if your pip does not automatically install for your Python 3.6+ distribution). To do this, enter the PERFORM root folder and execute

```
pip install -e .
```

This will install PERFORM, as well as any package dependencies which you have not yet installed. The only required package dependencies are numpy, scipy, and matplotlib. I have (arbitrarily) set a requirement of Python 3.6+, and no version requirements for the other dependencies. Eventually I'll thoroughly test version number requirements for all dependencies, but if you get an error related to these dependencies while running PERFORM, first try to upgrade them to the most recent versions. The code is only actively tested for Python 3.7 and Ubuntu 18.04; I make no guarantees that the code (particularly all plotting functions) works properly for macOS or Windows.

Once installation has completed without errors, a new script, PERFORM, will be added to your Python scripts. This is the command that you will use to execute PERFORM, and should always be followed by the path to the working directory of the case you would like to run, e.g.

```
perform /path/to/working/directory
```

The working directory of a case is dictated by the presence of a solverParams.inp **file**, which is described in Section 3. The code will not execute if there is not a properly-formatted solverParams.inp file in the specified working directory. **If you are running a ROM case, an additional** romParams.inp **file (described in Section 7) must also be placed in the working directory.** From this point on, the working directory will simply be referred to as $WORKDIR.

2.1 Input File Formatting

All input files (with the exception of initFiles and restart files) are ASCII files which are parsed using regular expressions. All input parameters must be formatted as inputName = inputValue, with as much white space before and after the equals sign as desired. A single line may contain a single input parameter definition, denoted by a single equals sign. An input file may contain as many blank lines as desired, and any line without an equals sign will be ignored (useful for user comments). Examples of various input parameters are given below

```
sampString      = "example/string_input"
sampInt         = 3
sampFloatDec    = 3.14159
sampFloatSci    = 6.02214e23
sampBool        = False
sampList        = [1.0, 2.0, 3.0]
sampListOfLists = [["1_1", "1_2"],["2_1", "2_2"]]
```

As a rule, you should write input values as if you were writing them directly in Python code. As seen above, string values should be enclosed in double quotes (single quotes is also valid) boolean values should be written precisely as `False` or `True` (case sensitive), lists should be enclosed by brackets (e.g. [val1, val2, val3]), and lists of lists should be formatted in kind (e.g. [[val11, val12],[val21, val22]]). Even if a list input only has one entry, it should be formatted as a list in the input file. Certain input parameters also accept a value of `None` (case sensitive). The `inputNames` are case sensitive, and string input parameters are also case sensitive. I am working on making these non-case sensitive where it's possible.

3 `solverParams.inp` Inputs

The input file `solverParams.inp` is the root input file from which the gas file, mesh file, and initial condition file are specified. Further, this file specifies all parameters related to the flux scheme, time discretization, robustness control, unsteady outputs, and visualizations. As mentioned previously, **it must be placed in the working directory, and must be named `solverParams.inp`**. Otherwise, the code will not function.

3.1 Parameter Types and Defaults

Parameter	Type	Default
`gasFile`	`str`	N/A
`meshFile`	`str`	N/A
`initFile`	`str`	N/A
`icParamsFile`	`str`	N/A
`dt`	`float`	N/A
`timeScheme`	`str`	N/A
`timeOrder`	`int`	N/A
`numSteps`	`int`	N/A
`subiterMax`	`int`	50
`resTol`	`float`	1.0e-12
`dualTime`	`bool`	`True`
`dtau`	`float`	1.0e-5
`adaptDTau`	`bool`	`False`
`CFL`	`float`	1.0
`VNN`	`float`	20.0
`runSteady`	`bool`	`False`
`steadyTol`	`float`	1.0e-12
`spaceScheme`	`str`	`"roe"`
`spaceOrder`	`int`	1
`gradLimiter`	`str`	`""`
`viscScheme`	`int`	0
`boundCond_inlet`	`str`	N/A
`press_inlet`	`float`	N/A
`vel_inlet`	`float`	N/A
`temp_inlet`	`float`	N/A
`rho_inlet`	`float`	N/A
`massFrac_inlet`	`list` of `float`	N/A
`pertType_inlet`	`float`	N/A
`pertPerc_inlet`	`float`	N/A
`pertFreq_inlet`	`list` of `float`	N/A
`boundCond_outlet`	`str`	N/A
`press_outlet`	`float`	N/A
`vel_outlet`	`float`	N/A

Parameter	Type	Default
temp_outlet	float	N/A
rho_outlet	float	N/A
massFrac_outlet	list of float	N/A
pertType_outlet	float	N/A
pertPerc_outlet	float	N/A
pertFreq_outlet	list of float	N/A
velAdd	float	0.0
resNormPrim	list of float	[1.0e5, 10.0, 300.0, 1.0]
sourceOn	bool	True
saveRestarts	bool	False
restartInterval	int	100
numRestarts	int	20
initFromRestarts	bool	False
probeLocs	list of float	[None]
probeVars	list of str	[None]]
outInterval	int	1
primOut	bool	True
consOut	bool	False
sourceOut	bool	False
RHSOut	bool	False
visInterval	int	1
visShow	bool	True
visSave	bool	False
visTypeX	str	N/A
visVarX	list of str	N/A
visXBoundsX	list of list of float	[[None,None]]
visYBoundsX	list of list of float	[[None,None]]
probeNumX	int	N/A
calcROM	bool	False

Table 1: solverParams.inp parameter types and defaults (where applicable).

3.2 Parameter Details

- gasFile: Absolute path to gas file.
- meshFile: Absolute path to mesh file.
- initFile: Absolute path to full initial primitive state profile (stored in a *.npy NumPy binary file) to initialize the unsteady solution from. If initFromRestart = True, this parameter will be ignored and the unsteady solution will be initialized from a restart file.
- icParamsFile: Absolute path to left/right (step function) primitive state parameters file to initialize the unsteady solution from. If initFile is set or initFromRestart = True, this parameter will be ignored.
- dt: Fixed time step size for numerical time integration.

- `timeScheme`: Name of numerical time integration scheme to use. Please see the solver documentation for details on each scheme.
  - Valid options: `"rkExp"`, `"bdf"`
- `timeOrder`: Order of accuracy for the chosen time integrator. Some time integrators have a fixed order of accuracy, while others may accept several different values. If a time integrator has a fixed order of accuracy and you have entered a different order of accuracy, a warning will display but execution will continue. If a time integrator accepts several values and an invalid value is entered, the solver will terminate with an error.
- `numSteps`: Number of discrete physical time steps to run the solver through. If the solver fails before this number is reached (either through a code failure or solution blowup), any unsteady output files will be dumped to disk with the suffix `"_FAILED"` appended to denote a failed solution.
- `subiterMax`: The maximum number of subiterations that the iterative solver for implicit time integration schemes may execute before concluding calculations for that physical time step.
- `resTol`: The threshold of convergence of the ℓ^2 norm of the Newton iteration residual below which the subiteration loop will automatically conclude.
- `dualTime`: Boolean flag to specify whether dual time-stepping should be used for an implicit time integration scheme.
- `dtau`: Fixed value of $\Delta\tau$ to use for dual time integration. Ignored if `adaptDTau = True`
- `adaptDTau`: Boolean flag to specify whether the value of $\Delta\tau$ should be adapted at each subiteration according to the below robustness control parameters.
- `CFL`: Dual time-stepping Courant–Friedrichs–Lewy number to adapt $\Delta\tau$ based on maximum wave speed in each cell. Smaller CFL numbers will result in smaller $\Delta\tau$, and greater regularization as a result.
- `VNN`: Dual time-stepping von Neumann number to adapt $\Delta\tau$ based on the mixture kinematic viscosity in each cell. Smaller VNN numbers will result in smaller $\Delta\tau$, and greater regularization as a result.
- `runSteady`: Boolean flag to specify whether to run the solver in "steady" mode. Refer to Section 3.3 for details.
- `spaceScheme`: Name of the numerical flux scheme to use. Please see the solver documentation for details on each scheme.
  - Valid options: `"roe"`
- `spaceOrder`: Order of accuracy of the state reconstructions at the cell faces for flux calculations. Must be a positive integer. If `spaceOrder = 1`, the cell-centered values are used. If `spaceOrder > 1`, finite difference stencils are used to compute cell-centered gradients, from which higher-order face reconstructions are computed. If the gradient calculation for the value entered has not been implemented, the solver with terminate with an error.
- `gradLimiter`: Name of the gradient limiter to use when computing higher-order face reconstructions. Please see the solver documentation for details on each scheme.

  - Valid options: "barth", "venkat"

- viscScheme: Integer flag to specify whether to compute viscous fluxes. If viscScheme = 0, the solver runs an inviscid calculation. If viscScheme = 1, viscous fluxes are applied. This will probably be changed to a Boolean flag at some point.

- boundCond_inlet: Name of the boundary condition to apply at the inlet. For details on each boundary condition, see the solver documentation. For required input parameters for a given boundary condition, see Section 3.4.1.

  - Valid options: "stagnation", "fullstate", "meanflow"

- press_inlet: Pressure-related value for inlet boundary condition calculations.

- vel_inlet: Velocity-related value for inlet boundary condition calculations.

- temp_inlet: Temperature-related value for inlet boundary condition calculations.

- rho_inlet: Density-related value for inlet boundary condition calculations.

- massFrac_inlet: Chemical composition-related value for inlet boundary condition calculations.

- pertType_inlet: Type of value to be perturbed at the inlet. See Section 3.4.1 for valid options for each boundCond_inlet value.

- pertPerc_inlet: Percentage of the specified perturbed value, determining the amplitude of the inlet perturbation signal. Should be entered in decimal format, e.g. for a 10% perturbation, enter pertPerc_inlet = 0.01. See Section 3.4.3 for more details.

- pertFreq_inlet: List of superimposed frequencies of the inlet perturbation. See Section 3.4.3 for more details.

- boundCond_outlet: Name of the boundary condition to apply at the outlet. For details on each boundary condition, see the solver documentation. For required input parameters for a given boundary condition, see Section 3.4.2.

- press_outlet: Pressure-related value for outlet boundary condition calculations.

- vel_outlet: Velocity-related value for outlet boundary condition calculations.

- temp_outlet: Temperature-related value for outlet boundary condition calculations.

- rho_outlet: Density-related value for outlet boundary condition calculations.

- massFrac_outlet: Chemical composition-related value for outlet boundary condition calculations.

- pertType_outlet: Type of value to be perturbed at the outlet. See Section 3.4.2 for valid options for each boundCond_outlet value.

  - "subsonic", "meanflow"

- `pertPerc_outlet`: Percentage of the specified perturbed value, determining the amplitude of the outlet perturbation signal. Should be entered in decimal format, e.g. for a 10% perturbation, enter `pertPerc_inlet = 0.1`. See Section 3.4.3 for more details.
- `pertFreq_outlet`: List of superimposed frequencies of the outlet perturbation. See Section 3.4.3 for more details.
- `velAdd`: Velocity to be added to the entire initial condition velocity field. Accepts negative values.
- `resNormPrim`: List of values by which to normalize each field of the ℓ^2 and ℓ^1 primitive state residual norms (for dual time-stepping) before averaging across all fields. They are order by pressure, velocity, temperature, and then all species mass fractions except the last. This ensures that the norms of each residual field contribute roughly equally to the average norm used to determine Newton's method convergence.
- `sourceOn`: Boolean flag to specify whether to apply the reaction source term. This is `True` by default; setting it manually to `False` turns off the source term. This can save computational cost for non-reactive cases.
- `saveRestarts`: Boolean flag to specify whether to save restart files.
- `restartInterval`: Physical time step interval at which to save restart files.
- `numRestarts`: Maximum number of restart files to store. After this threshold has been reached, the count returns to 1 and the first restart file is overwritten by the next restart file (and so on).
- `initFromRestarts`: Boolean flag to determine whether to initialize the unsteady solution from
- `probeLocs`: List of locations in the spatial domain to place point monitors. The probe measures values at the cell center closest to the specified location. If a location is less than the inlet boundary location, the inlet ghost cell will be monitored. Likewise, if a location is greater than the outlet boundary location, the outlet ghost cell will be monitored. These probe monitors are recorded at every physical time iteration and the time history is written to disk. See Section 8.2 for more details on the output.
- `probeVars`: A list of fields to be probed at each specified probe location.
  - Valid for all probes: `"pressure"`, `"velocity"`, `"temperature"`, `"density"`, `"momentum"`, `"energy"`, `"species"X`, `"density-species"X` (where `X` is replaced by the integer number of the desired chemical species to be probed, e.g. `"species2"` for the second species specified in the gas file).
  - Valid options for interior probes only: `"source"`
- `outInterval`: Physical time step interval at which to save unsteady field data.
- `primOut`: Boolean flag to specify whether the unsteady primitive field should be saved.
- `consOut`: Boolean flag to specify whether the unsteady conservative field should be saved.
- `sourceOut`: Boolean flag to specify whether the unsteady source term field should be saved.
- `RHSOut`: Boolean flag to specify whether the unsteady right-hand-side field should be saved.

- `visInterval`: Physical time step interval at which to draw any requested field/probe plots. If no plots are requested, this parameter is ignored.
- `visShow`: Boolean flag to specify whether field/probe plots should be displayed on the user's monitor at the interval specified by `visInterval`. If no plots are requested, this parameter is ignored.
- `visSave`: Boolean flag to specify whether field/probe plots should be saved to disk at the interval specified by `visInterval`. If no plots are requested, this parameter is ignored. See Section 8.4 for more details.
- `visTypeX`: Type of data to visualize in the `X`-th figure. For example, `visType3` would specify the type of the third plot to be visualized. Values of `X` must start from 1 and progress by one for each subsequent plot. Any gap in these numbers will cause any plots after the break to be ignored (e.g. specifying `visType1`, `visType3`, and `visType4` without specifying `visType2` will automatically ignore the plots for `visType3` and `visType4`).
  - Valid options: `"field"`, `"probe"`
- `probeNumX`: 1-indexed number of the point monitor to visualize in the `X`-th figure if `visTypeX = "probe"`. Must correspond to a valid probe number.
- `visVarX`: A list of fields to be plotted in the `X`-th figure. Note that for `visTypeX= "probe"` figures, if a specified field is not being monitored at the probe specified by `probeNumX`, the solver will terminate with an error.
  - Valid for all plots: `"pressure"`, `"velocity"`, `"temperature"`, `"density"`, `"momentum"`, `"energy"`, `"species"X`, `"density-species"X` (where `X` is replaced by the integer number of the desired chemical species to be probed, e.g. `"species2"` for the second species specified in the gas file).
- `visXBoundsX`: List of lists, where each sub-list corresponds to the plots specified in `visVarX`. Each sublist contains two entries corresponding the lower and upper y-axis bounds for visualization of `visVarX`.
- `visYBoundsX`: List of lists, where each sub-list corresponds to the plots specified in `visVarX`. Each sublist contains two entries corresponding the lower and upper x-axis bounds for visualization of `visVarX`.
- `calcROM`: Boolean flag to specify whether to run a ROM simulation. If set to `True`, a `romParams.inp` file must also be placed in the working directory. See Section 7 for more details on this input file.

3.3 Running in "Steady" Mode

Setting the Boolean flag `runSteady = True` slightly alters the solver behavior to run in a sort of "steady-state solver" mode. To be completely clear, `PERFORM` is an unsteady solver and there are no true steady solutions for the types of problems it is designed to simulate. However, this "steady" mode is designed specifically for solving flame simulations in which bulk advection is carefully balanced with

chemical diffusion and reaction forces, resulting in a roughly stationary flame, which is as close to a steady solution as one can expect for these cases. This stationary flame acts as a good "mean" flow for stationary flame problems with external forcing, or as an initial condition for transient flame problems (combined with a non-zero `velAdd`).

The exact changes in behavior are as follows:

1. The ℓ^2 and ℓ^1 norms displayed in the terminal is the norm of the change in the primitive state between physical time steps. This is opposed to no residual output for explicit time integration schemes, or the linear solve residual norm for implicit time integration schemes.
2. The time history of the above residual norm will be written to the file `$WORKDIR/UnsteadyFieldResults/steadyConvergence.dat`.
3. The solver will terminate early if the ℓ^2 norm of the solution change converges below the tolerance set by `steadyTol`.

This "steady" solver can be run for both explicit and implicit time integration schemes. The procedure for obtaining steady flame solutions is incredibly tedious, generally requiring carefully manually tuning the boundary conditions to achieve a certain inlet velocity until a point at which the advection downstream is balanced with the diffusion and reaction moving upstream. During this tuning procedure, the user often must visually confirm that the flame is not moving by watching the field plots closely. Again, this process is incredibly tedious, but the "steady" solver helps facilitate this by providing at least one quantitative metric for determining if a steady flame solution has been achieved.

3.4 Boundary Condition Parameters

As mentioned in the above parameter details, the requirements and interpretations of the `*_inlet` and `*_outlet` parameters depends on the boundary condition specified by `boundCond_inlet` / `boundCond_outlet`. Below, we specify the required parameters and their interpretations for each respective boundary condition. Valid options of `pertType` for each boundary condition are also specified. If an invalid entry for `pertType` is supplied, it will simply be ignored. For the mathematical definitions of these boundary conditions, please refer to the `PERFORM` solver documentation. This section will be updated as new boundary conditions are implemented.

3.4.1 Inlet Boundary Conditions and Parameters

1. `"stagnation"` (specify stagnation pressure and temperature)
   - `press_inlet`: Specified stagnation pressure at the inlet.
   - `temp_inlet`: Specified stagnation temperature at the inlet.
   - `massFrac_inlet`: Fixed mixture composition at the inlet.
2. `"fullstate"` (specify full primitive state)
   - `press_inlet`: Fixed static pressure at the inlet.
   - `vel_inlet`: Fixed velocity at the inlet.

- `temp_inlet`: Fixed static temperature at the inlet.
- `massFrac_inlet`: Fixed mixture composition at the inlet
- `pertType_inlet` (optional): Accepts `"pressure"`, `"velocity"`, or `"temperature"` to perturb the values of the appropriate fixed quantity.

3. `"meanflow"` (non-reflective boundary)

   - `press_inlet`: Specified mean upstream static pressure.
   - `temp_inlet`: Specified mean upstream static temperature.
   - `massFrac_inlet`: Fixed mixture composition at the inlet.
   - `vel_inlet`: Specified mean upstream value of ρc, where c is the sound speed.
   - `rho_inlet`: Specified mean upstream value of ρc_p, where c_p is the specific heat capacity at constant pressure.
   - `pertType_inlet` (optional): Accepts `"pressure"` to perturb the mean upstream pressure.

3.4.2 Outlet Boundary Conditions and Parameters

1. `"subsonic"` (specify static pressure)

   - `press_outlet`: Specified static pressure at the outlet.
   - `massFrac_outlet`: Fixed mixture composition at the outlet.
   - `pertType_outlet` (optional): Accepts `"pressure"` to perturb the pressure at the outlet.

2. `"meanflow"` (non-reflective boundary)

   - `press_outlet`: Specified mean downstream static pressure.
   - `vel_outlet`: Specified mean downstream value of ρc, where c is the sound speed.
   - `rho_outlet`: Specified mean downstream value of ρc_p, where c_p is the specific heat capacity at constant pressure.
   - `pertType_outlet` (optional): Accepts `"pressure"` to perturb the mean downstream pressure.

3.4.3 Boundary Condition Perturbations

Setting valid values for `pertType_inlet` or `pertType_outlet`, as well as non-zero values of `pertPerc_inlet` / `pertFreq_inlet` or `pertPerc_outlet` / `pertFreq_outlet`, initiates external forcing at the appropriate boundary. The perturbation signal is a simple sinusoid, given for a given perturbed quantity α in the boundary ghost cell as

$$\alpha(t) = \overline{\alpha}\left(1 + A\sum_{i=1}^{N_f}\sin(2\pi f_i t)\right)$$

where $\overline{\alpha}$ is the relevant reference quantity given in `solverParams.inp`, f_i are the signal frequencies in `pertFreq_inlet`/`pertFreq_outlet`, and A is the amplitude percentage `pertPerc_inlet`/`pertPerc_outlet`. For example, if the user sets (among other required parameters)

```
boundCond_outlet = "meanflow"
press_outlet = 1.0e6
pertType_outlet = "pressure"
pertPerc_outlet = 0.05
pertFreq = [2000.0, 5000.0]
```

this will result in two perturbation signals (one of 2 kHz, another of 5 kHz) of the mean downstream static pressure with amplitude 50 kPa.

4 Gas File Inputs

The gas file defines all properties of the various chemical species modeled in a given simulation, along with all parameters which define the reactions between these species. The name and placement of this file are arbitrary, as it is referenced from the `gasFile` parameter given in `solverParams.inp`. The parameters for describing the finite-rate reactions are given in Section 4.1. The parameters for describing the chemical the calorically-perfect gas model and their details are given in Section 4.2. To be abundantly clear, **these parameters should all be given in the same file**, but they are split into different sections here for clarity. As a warning, the format of this file may change significantly as it is streamlined to accommodate other gas models such as thermally-perfect gases and reaction modeling via Cantera.

4.1 Finite-rate Kinetics

4.1.1 Parameter Types and Defaults

Parameter	Type	Default
`nu`	`list` of `float`	N/A
`nuArr`	`list` of `float`	N/A
`actEnergy`	`float`	N/A
`preExpFact`	`float`	N/A

Table 2: Gas file reaction parameter types and defaults (where applicable).

4.1.2 Parameter Details

- `nu`: Global reaction stoichiometric coefficients. Reactants should have positive values, while products should have negative values.
- `nuArr`: Global reaction molar concentration exponents for all chemical species. Those chemical species that don't participate in the reaction should just be assigned a value of 0.0.
- `actEnergy`: Arrhenius exponential factor $-E_a/R_u$, where E_a is the actual activation energy, and R_u is the universal gas constant. **Please note the negative factor**.
- `preExpFact`: Arrhenius pre-exponential factor

4.2 Calorically Perfect Gas Model

4.2.1 Parameter Types and Defaults

Parameter	Type	Default
`gasType`	`str`	`"cpg"`
`numSpeciesFull`	`int`	N/A
`molWeights`	`list` of `float`	N/A
`enthRef`	`list` of `float`	N/A
`Cp`	`list` of `float`	N/A
`Pr`	`list` of `float`	N/A
`Sc`	`list` of `float`	N/A
`tempRef`	`list` of `float`	N/A
`muRef`	`list` of `float`	N/A

Table 3: Gas file CPG parameter types and defaults (where applicable).

4.2.2 Parameter Details

- `gasType`: Name of gas model to be used.
  - Valid options: `"cpg"`
- `numSpeciesFull`: Total number of species participating in simulation.
- `molWeights`: Molecular weights of each species. Must have `numSpeciesFull` entries.
- `enthRef`: Reference enthalpy at 0 K of each species. Must have `numSpeciesFull` entries.
- `Cp`: Constant specific heat capacity at constant pressure for each species. Must have `numSpeciesFull` entries.
- `Pr`: Prandtl number of each species. Must have `numSpeciesFull` entries.
- `Sc`: Schmidt number of each species. Must have `numSpeciesFull` entries.
- `muRef`: Reference dynamic viscosity of each species for Sutherland's law. Must have `numSpeciesFull` entries.
- `tempRef`: Reference temperature of each species for Sutherland's law. If `tempRef[i] = 0` for any species, it will be assumed that its dynamic viscosity is constant and equal to `muRef[i]`. Must have `numSpeciesFull` entries.

5 Mesh File Inputs

The mesh file is incredibly simple at the moment, as `PERFORM` can only accommodate uniform meshes. For now, it simply defines the left and right boundary coordinates, along with the number of finite volume cells. As with the gas file, the name and placement of this file is arbitrary, as it is referenced from the `meshFile` parameter in `solverParams.inp`. Hopefully, the code will be able to at least accommodate non-uniform meshes, and perhaps even some sort of rudimentary mesh adaptation. Below are the required parameters for defining the computational mesh and their details.

5.1 Parameter Types and Defaults

Parameter	Type	Default
`xL`	`float`	N/A
`xR`	`float`	N/A
`numCells`	`int`	N/A

Table 4: Mesh file parameter types and defaults (where applicable).

5.2 Parameter Details

- `xL`: Left-most boundary coordinate of the spatial domain. This point will be the coordinate of the left face of the left-most finite volume cell.
- `xR`: Right-most boundary coordinate of the spatial domain. This point will be the coordinate of the right face of the right-most finite volume cell.
- `numCells`: Total number of finite volume cells in the discretized spatial domain.

6 Initial Conditions Inputs

Unsteady solutions can be initialized in three different ways in PERFORM: full primitive state profiles (Section 6.2), piecewise uniform function parameters files (Section 6.1), or restart files (Section 6.3). The methods for generating each is detailed below.

6.1 Piecewise Uniform Initial Condition Input File

A file to generate a piecewise uniform initial condition (located by the icParamsFile parameter in solverParams.inp) defines a "left" and "right" state of the initial condition. One may think of this as a sort of "shock tube" situation, in which two chambers of gas are separated by a diaphragm that is burst at the beginning of the simulation. The full primitive state must be defined in both chambers. The parameters are given below, along with their details.

As noted previously, setting initFile (full primitive state profile) or initFromRestart = True (restart file) will override any setting of icParamsFile.

6.1.1 Parameter Types and Defaults

Parameter	Type	Default
xSplit	float	N/A
pressLeft	float	N/A
velLeft	float	N/A
tempLeft	float	N/A
massFracLeft	list of float	N/A
pressRight	float	N/A
velLeft	float	N/A
tempLeft	float	N/A
massFracLeft	list of float	N/A

Table 5: icParamFile parameter types and defaults (where applicable).

6.1.2 Parameter Details

- xSplit: Location in spatial domain at which the piecewise uniform solution will be split. All cell centers with coordinates less than this value will be assigned to the "left" state, and those with coordinates greater than this value will be assigned to the "right" state.
- pressLeft: Static pressure in "left" state.
- velLeft: Velocity in "left" state.
- tempLeft: Temperature in "left" state.
- massFracLeft: Species mass fractions in "left" state. Must contain numSpeciesFull elements, and they must sum to 1.0.

- `pressRight`: Static pressure in "right" state.
- `velRight`: Velocity in "right" state.
- `tempRight`: Temperature in "right" state.
- `massFracRight`: Species mass fractions in "right" state. Must contain `numSpeciesFull` elements, and they must sum to 1.0.

6.2 `initFile` Input File

Providing a complete primitive state profile is by far the simplest method available. The `initFile` parameter in `solverParams.inp` provides the arbitrary location of a NumPy binary (`*.npy`) containing a single NumPy array. This NumPy array must be a two-dimensional array, where the first dimension is the number of governing equations in the system (3 + `numSpeciesFull` - 1) and the second dimension is the number of cells in the discretized spatial domain. The order of the first dimension *must* be ordered by pressure, velocity, temperature, and then chemical species mass fraction. The chemical species mass fractions must be ordered as they are in the gas file. This file can be generated however you like, such as ripping it manually from the unsteady outputs of a past `PERFORM` run, or generating a more complex series of discontinuous steps than what the `icParamsFile.inp` settings handle natively.

As noted previously, setting `initFile` will override any setting of `icParamsFile`. Setting `initFromRestart = True` (initializing from restart file) will override any setting of `initFile`.

6.3 Restart Files

Restart files accomplish what the name implies: restarting the simulation from a previous point in the simulation. Restart files are saved to `$WORKDIR/RestartFiles` when `saveRestarts = True` at an interval specified by `restartInterval` in `solverParams.inp`. Two files are saved to reference a restart solution: a `restartIter.dat` file and a `restartFile_X.npz` file, where `X` is the *restart iteration number*. The latter file contains both the primitive solution saved at that restart iteration, as well as the physical solution time associated with that solution. The former file is an ASCII file containing the restart iteration number of the most recently-written restart file, and thus points to which `restartFile_X.npz` should be read in to initialize the solution. It is overwritten every time a restart file is written. Similarly, the maximum number of `restartFile_X.npz` saved to disk is dictated by `numRestarts`. When this threshold is reached, the restart iteration number will loop back to 1 and begin overwriting old restart files.

Setting `initFromRestart = True` will initialize the solution from the restart file whose restart iteration number matches the one given in `restartIter.dat`. Thus, without modification, the solution will restart from the most recently generated restart file. However, if the user wishes to restart from a different iteration number, they may manually change the iteration number stored in `restartIter.dat`.

As noted previously, setting `initFromRestart = True` will override any setting of both `icParamsFile` and `initFile`.

7 `romParams.inp` Input

This section will be updated shortly, check back soon!

8 Outputs

This section outlines where the various outputs of PERFORM are written, as well as their formats. All output directories referenced here are automatically generated in the working directory

8.1 Unsteady Field Data Outputs

Unsteady field data represents the time evolution of an unsteady field at the time step iteration interval specified by outInterval in solverParams.inp. All unsteady field data is written to $WORKDIR/UnsteadyFieldResults, and currently comes in four flavors: primitive state output, conservative state output, RHS term output, and source term output. All have the same general form: a NumPy binary file containing a single NumPy array with three dimensions. The first dimension is the number of variables, the second is the number of cells in the computational mesh, and the third is the number of time steps saved in the output file. The primitive state, conservative state, and RHS term output have the same number of variables, equal to the number of governing equations (3+numSpeciesFull−1), while the source term only has numSpeciesFull−1 variables.

Primitive state field data is saved if primOut = True and has the prefix solPrim_*. Conservative state field data is saved if consOut = True and has the prefix solCons_*. RHS term field data is saved if RHSOut = True and has the prefix solRHS_*. Source term field data is saved if sourceOut = True and has the prefix source_*.

Unsteady field data may have three different main suffixes, depending on solver parameters: *_FOM for an unsteady full-order model simulation, *_steady for a "steady" full-order model simulation (see Section 3.3 for details), or *_ROM for a reduced-order model simulation. An additional suffix, *_FAILED, is appended the this if the solver fails (code error or solution blowup). Thus, the conservative field results for a failed ROM run would have the name solCons_ROM_FAILED.npy, while a successful run would simply generate solCons_ROM.npy.

8.2 Probe Monitor Data Outputs

Probe/point monitor data represents the time evolution of an unsteady field variable at a single finite volume cell. Probe locations are specified by probeLocs in solverParams.inp, and the fields to be measured are specified by probeVars. Valid options for probeVars are listed in Section 3. Probe measurements are taken at every single physical time step. Data for each probe is saved to a separate file in $WORKDIR/ProbeResults. Each has the format of a NumPy binary file containing a single two-dimensional NumPy array. The first dimension is the number of variables listed in probeVars plus one, as the physical solution time is also stored at each iteration. The second dimension is the number of physical time steps for which the simulation was run.

The name of each probe begins with probe_*, and is followed by a list of the variables stored in the probe data. Finally, the same suffixes mentioned in Section 8.1 are applied depending on the solver settings: *_FOM, *_steady, and *_ROM. Again, if the solver fails, the suffix *_FAILED will also be appended. Finally, the 1-indexed number of a the probe will be appended to the end of the file. For example, the second probe monitoring the velocity and momentum of a "steady" solve which fails will have the file name probe_velocity_momentum_2_steady_FAILED.npy.

8.3 Restart File Outputs

All restart file data is stored in `$WORKDIR/RestartFiles`. Please refer to Section 6.3 for details on the formatting and contents of restart files.

8.4 Plot Image Outputs

During simulation runtime, `PERFORM` is capable of generating two types of plots via `matplotlib`: field plots and probe monitor plots. If `visShow = True` in `solverParams.inp`, then these images are displayed on the user's monitor. If `visSave = True`, they are saved to disk. The interval of displaying/saving the figures is given by `visInterval`. Each figure corresponds to a single instance of `visTypeX`, within which there may be several plots. Each probe figure corresponds to a single probe, from which multiple probed variables may be extracted.

The probe plots display the entire time history of the probed data, with the probed variable data plotted on the y-axis and time plotten on the x-axis. Field plots, on the other hand, display instantaneous snapshots of the entire field with the field data plotted on the y-axis and cell center coordinates plotted on the x-axis.

All saved images are PNG images stored within `$WORKDIR/ImageResults`. The names of probe plots follow the same patters given to the probe data files (except with the file extension `*.png`, of course). A single figure is generated for a given probe figure. Field plots, on the other hand, save an instantaneous snapshot of the field plots at the interval set by `visInterval`. These are stored within a subdirectory following the same pattern given to field data files, except the prefix of the file is given by `$WORKDIR/ImageResults/field_*`. Within this subdirectory, individual images have the prefix `fig_*`, followed by the number of the image in the series of expected image numbers to be generated by a given run. Of course, if a simulation terminates early any field plots that were expected to be generated will not be generated.

An example of an instantaneous field figure for the Sod shock tube example (Section 9.1) and an accompanying probe figure are given in Figs.

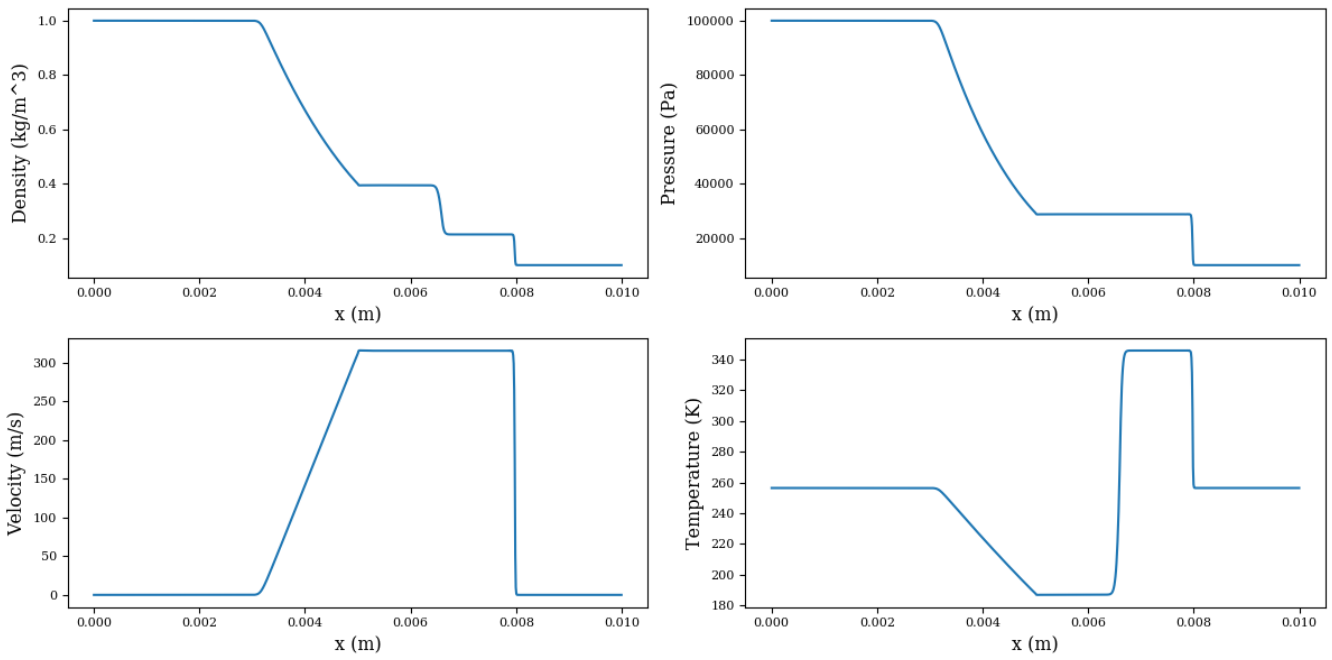

Figure 1: Sample field plots.

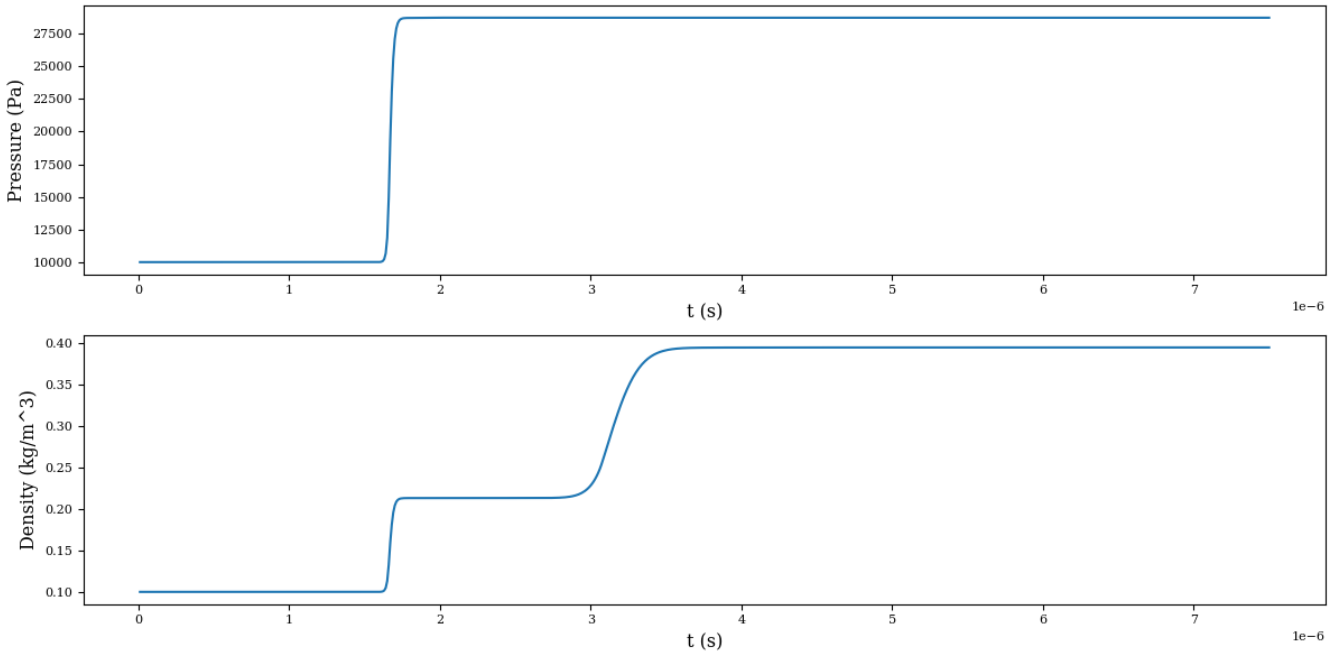

Figure 2: Sample probe plots.

9 Example Cases

This section will be updated shortly, check back soon!

9.1 Sod Shock Tube

9.2 Contact Surface

9.3 Transient Flame

10 Utility Scripts

This section will be updated shortly, check back soon!