Laboratory of Computational Plasma Physics

CRANE User Guide

Contents

Chapter 1

Introduction

1.1 CRANE (Chemical ReAction Network)

This documentation is intended to give a brief overview of how to run *Crane*, is an open source software dedicated to modeling the time evolution of chemically reactive plasma species. The mathematical kernel of the problem is a system of rate equations:

$$\frac{d[n_i]}{dt} = \sum_{j=1}^{j_{max}} S_{ij} \tag{1.1}$$

For a reaction of the form:

$$a_1 A + bB \rightarrow a_2 A + cC \tag{1.2}$$

the rate of production of each species, S_{ij}, is defined as:

$$\begin{aligned} S_A &= (a_2 - a_1) K_j (n_A)_1^a (n_B)^b \\ S_B &= -b K_j (n_A)_1^a (n_B)^b \\ S_C &= c K_j (n_A)_1^a (n_B)^b \end{aligned}$$

By design, *Crane* will read equations written in the form of equation 1.2 and automatically construct the necessary source and sink terms for each species. In this way, large reaction networks may be written in human-readable form.

Crane was written as part of the MOOSE finite element framework, and as such it is a fully open-source software that may be downloaded from the Laboratory of Computational Physics GitHub repository. The software is also distributed as freeware as a DMG (Mac) if the user does not wish to contribute to the project; however, please note that the freeware version may not include all features available by using the compiled source code. (For questions about this, please contact the developers or visit the FAQ on the LCPP website.)

1.2 Source Code

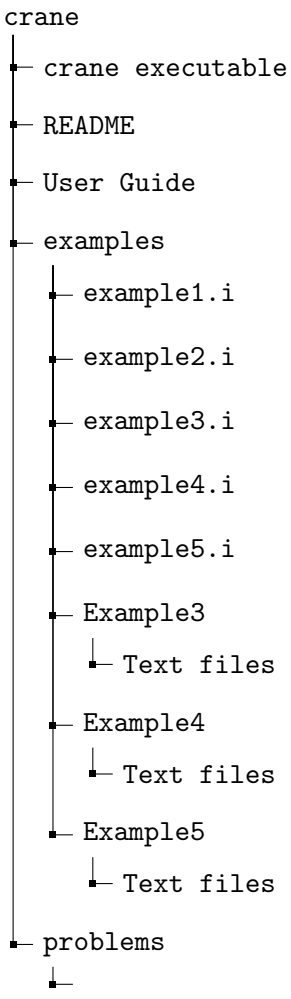
The source code is available at the UIUC Laboratory for Computational Physics GitHub repository. Note that in order to compile the source code, the user must first follow the MOOSE framework installation instructions.

1.3 Precompiled Download

For convenience, *Crane* is also available as a pre-compiled executable. Some features may be missing from the standalone release, namely the ability to couple to other MOOSE applications. However, the basic capability of solving a system of reaction equations of arbitrary size in zero dimensions remains in this version.

Precompiled Directory Structure

The precompiled version of *CRANE* includes the executable, license information, and example problems along with an empty `problems` directory. The file structure is laid out as follows:



In both the source code and precompiled versions, it is recommended that the user develops new input files in the `problems` directory.

Chapter 2

Input File Structure

As *Crane* was developed as part of the MOOSE framework, input files follow the same input file syntax as all MOOSE applications. Information about MOOSE input files in general may be found on the MOOSE framework website. This section outlines all of the necessary input file features needed to run a simulation, including features that are unique to *Crane*.

A basic MOOSE input file requires five sections, called "blocks" in MOOSE parlance, the names of which are enclosed in brackets (e.g. [Mesh]): [Mesh], [Variables], [Kernels], [Executioner], and [Outputs]. After the options are filled in, each block is then closed with empty brackets (e.g. []). A block unique to *Crane* named [ChemicalReactions] was developed to add a system of reactions to the problem. Available blocks are listed below.

The pound symbol, #, may be used to denote comments in the input file. Comments will be used to highlight features of each block in the snippets shown throughout this user guide.

Blocks	Definitions
[Mesh]	The numerical mesh definition. MOOSE may automatically generate a mesh or use an external mesh (produced by software like Gmsh). In 0D, the mesh should be a single element.
[Variables]	The list of nonlinear variables. In *Crane*, these should be the chemical species, including electrons.
[AuxVariables]	The list of auxiliary variables. These variables are used for values that are needed for the solution or are a desired output, but are not a chemical species. (For example, electron mobility or ionization fraction are two commonly used AuxVariables.)
[Kernels]	In MOOSE, a "Kernel" is a single term in the system of equations that is being solved. Each of the production terms shown in section 1.1 is a Kernel. *Crane* automatically adds the necessary source/sink terms; the user must only add time derivative terms for each species.
[ScalarKernels]	Same as Kernels, but applied to ScalarVariables. These are almost always used in *Crane* simulations unless the software is being coupled to another MOOSE application.
[AuxKernels]	These are used to compute the value of the AuxVariable defined in the AuxVariable block. They are typically calculated at the beginning or end of each timestep, although they may be computed as frequently as every linear iteration.
[AuxScalarKernels]	Same as AuxKernels, but applied to scalar AuxVariables.
[ChemicalReactions]	Defines the list of reactions that must be solved. Chapter 3 will explain all available options in detail.
[Executioner]	Defines the numerical solution method, end time, timestep information, and any preconditioning that must be applied.
[Outputs]	Defines the outputs that should be produced. Typical output format in 0D for *Crane* would be a CSV file.

The following sections detail the relevant options in each block, and a sample input file is constructed at the end of this chapter. Please note that with the exception of some specific AuxKernels and the entire ChemicalReactions block, all of the blocks listed above are typical of MOOSE input files. For more detailed information please visit the MOOSE framework website.

2.1 Mesh

The Mesh block defines the numerical mesh that the MOOSE framework will operate on. In general, MOOSE supports meshes from 1D-3D. However, *Crane* by itself is intended to be used as a 0D reaction network solver, so the Mesh in this case is considered a "dummy" mesh that is not used. In this case the Mesh block will be a 1D mesh consisting of a single element:

```
[Mesh]
    # A GeneratedMesh should always be used with Crane.
    type = GeneratedMesh
    dim = 1
    xmin = 0
    xmax = 1
    nx = 1
    # The "mesh" will extend from 0 to 1 in 1D with a single element.
[]
```

2.2 Variables

As a standalone solver, *Crane* will be run in 0D. In this case `SCALAR` variables should be used. Within the Variables block each variable must be individually named and its initial condition must be set. If an initial condition is not set, the value will default to 1. The following Variables block defines two variables named `variable1` and `variable2` with initial conditions 10 and 0, respectively.

```
[Variables]
    # First order variables should always be used with Crane.
    [./variable1] # This is the variable's name.
        family = SCALAR
        order = FIRST
        initial_condition = 10
    [../]

    [./variable2]
        family = SCALAR
        order = FIRST
        initial_condition = 0
    [../]
[]
```

The MOOSE framework includes several different families and orders to choose from. However, it is recommended that when using *Crane* as a standalone solver, the user uses `SCALAR`, `FIRST` order variables. Note that for user readability it is convenient if the variable names are defined in a human-readable format; e.g. if the user is solving for electrons and argon ions, the variables may be named `e` and `Ar+`.

NOTICE

Do **NOT** use carot symbols in variable names, e.g. $Ar\hat{}+$. Such a name may be accepted by MOOSE, but if the variable is used in any parsed function in the simulation (in an equation-based rate coefficient or auxiliary kernel, for example) **the carot will be read as an operator and will not be recognized as part of a variable name**.

2.3 Kernels and ScalarKernels

Kernels define a single piece of physics. In terms of a system of equations, a single Kernel/ScalarKernel represents an individual term in the user's partial differential equations (e.g. each source and sink term for every variable is a Kernel/ScalarKernel, as is each time derivative term).

Crane was developed such that the user must only input time derivative Kernels/ScalarKernels for their intended problem. The `ChemicalReactions` block will automatically add all source and sink terms to the problem. For the variables shown in the previous section, time derivative terms may be added like so:

```
[ScalarKernels]
  [./variable1_dt]
    type = ODETimeDerivative
    variable = variable1  # Same name as declared above
  [../]

  [./variable2_dt]
    type = ODETimeDerivative
    variable = variable2
  [../]
[]
```

The block shown above will construct two ScalarKernels named `variable1_dt` and `variable2_dt`. Both ScalarKernels are of type `ODETimeDerivative`, which means that the term will be of the form $\frac{\partial dx}{\partial t}$, where x is the variable (`variable1` and `variable2`). Note that the name of the `ScalarKernel` is not important; it is only used by the code itself and may be named anything convenient to the user. The names must be unique, however.

For Kernels/ScalarKernels, the `type` parameter defines the specific Kernel that should be applied. For ScalarKernels a time derivative term is defined by the `ODETimeDerivative` class, and for Kernels it is defined by the `TimeDerivative` class. **Time derivatives must be added manually in the current version of *Crane*.** No other Kernels/ScalarKernels must be added manually to solve a system of chemical reactions.

2.4 AuxVariables and AuxKernels/AuxScalarKernels

Auxiliary variables are values that are either (a) necessary as input to Kernels or (b) are useful quantities that the user may want to have computed by the program, but are not themselves nonlinear variables. For example, rate coefficients are added to the system as scalar AuxVariables and are passed to the source and sink term ScalarKernels. As an example of the second type, the user may want to have the ionization fraction calculated as a function of time. The ionization fraction depends on nonlinear variables (namely the total gas density and total density of all ionized species), but does not contribute to the solution of those variables.

Auxiliary kernels provide a value to those auxiliary variables through some calculation. If the total number of ionized particles was an AuxVariable named `total_ionized`, the AuxKernel would calculate the sum of all the ionized species and apply that value to the `total_ionized` variable. The user may also read data from external files with AuxKernels/AuxScalarKernels. This is useful if the user has a tabulated set of experimental data representing a quantity such as electron temperature as a function of time.

A full list of available AuxKernels/AuxScalarKernels are provided in the appendix. An example of reading a scalar AuxVariable from an external data file is shown below. In this example, the user has a file of electron mobility values tabulated as a function of reduced electric field. In order to read those values, first the user must declare two AuxVariables: (1) a mobility variable to store the values, and (2) a reduced electric field variable to sample the correct mobility value. Then an AuxScalarKernel must be applied that declares the variable that is being computed, the variable that will be used as the "sampler", and the location of the data file. The `DataReadScalar` class may be used to accomplish this task.

```
# First an AuxVariable must be declared.
# They are declared in the same way as normal variables, but
# within an AuxVariable block.
[AuxVariables]
  [./reduced_field]
    order = FIRST
    family = SCALAR
    initial_condition = 7.7667949e-20
  [../]

  [./mobility]
    order = FIRST
    family = SCALAR
  [../]
[]

# Note that the mobility AuxVariable was not given an initial
# condition. However, the AuxScalarKernel may be executed on
# INITIAL,  providing the initial value for mobility.

[AuxScalarKernels]
  [./mobility_calculation]
    type = DataReadScalar
    variable = mobility  # Same AuxVariable declared above.
    sampler = reduced_field
    property_file = 'ExampleFiles/electron_mobility.txt'
    execute_on = 'INITIAL TIMESTEP_BEGIN'
  [../]
[]
```

The above input file snippet will create the two necessary variables, read the reduced field versus mobility data from the 'electron_mobility.txt' data file (which is located in the `ExampleFiles` directory), and use the reduced field variable to sample the correct mobility value from the file. Note that the `DataReadScalar` class will perform a spline interpolation to calculate the value. The `execute_on` parameter dictates when an AuxScalarKernel is going to be run. In this case, it is set to both `INITIAL` and `TIMESTEP_BEGIN`, which means the AuxScalarKernel will be executed at runtime *and* at the beginning of every subsequent timestep.

2.5 Executioner, Outputs, and Preconditioning

The `Executioner` block dictates the parameters of the solve - timestep information, runtime, the numerical schemes, checks for steady state, and adaptive timestepping may all be modified and applied in the executioner. Two examples are shown below.

Example 1: Simple Executioner

```
[Executioner]
  type = Transient
  start_time = 0
  end_time = 50
  dt = 0.1
  solve_type = NEWTON
[]
```

In Example 1, a `Transient` problem is defined starting at time $t = 0$ and ending at time $t = 50$, with timesteps of 0.1. The numerical scheme is defined as the Newton method. **The Newton method is recommended for all solves in *Crane*.** If a problem will not converge it is suggested to use either the preconditioned jacobian free Newton-Krylov solver (`solve_type = PJFNK`) or the jacobian free Newton-Krylov solver (`solve_type = JFNK`). The Newton solver should be optimal for most reaction network problems, however.

More advanced options include checking for steady state and applying an adaptive timestepper, which will modify the timestep as the solution runs based on whether or not the convergence criteria are being met (e.g. if convergence is met without issue, the timestep will increase by a set amount. If convergence fails, the timestep will be dialed back.)

Example 2: Adaptive Timestepping

```
[Executioner]
  type = Transient
  end_time = 1e-3
  solve_type = NEWTON
  dtmin = 1e-12
  dtmax = 1e-4
  steady_state_detection = true
  steady_state_start_time = 1e-4
  [./TimeStepper]
    type = IterationAdaptiveDT
    cutback_factor = 0.9
    dt = 1e-10
    growth_factor = 1.01
  [../]
[]
```

In Example 2, the end time of the simulation is set to $1e-3$ seconds. We have also set a parameter called `steady_state_detection` to `true`, and the start time for that steady state detection was set to $1e-4$. This means that once the simulation reaches $t = 1e-4$ it will begin checking to see if the solution has reached steady state, and if it has it will terminate the simulation.

A subblock named `TimeStepper` has also been declared, with a type of `IterationAdaptiveDT`. This will indicate to *Crane* that an adaptive timestep should be utilized. The timestep will start with a value of $dt = 1e-10$, and at each timestep it will either grow by a factor of 1.01 or be reduced by a factor of 0.9 depending on whether or not convergence has been achieved. In addition, a minimum timestep of $1e-12$ and a maximum timestep of $1e-4$ has been set; if convergence fails enough times that the timestep is scaled back to $1e-12$ or below, the simulation will be terminated. Similarly, the timestep cannot grow past $dt = 1e-4$ even if convergence continues to be met.

Adaptive timestepping is useful for large problems because it has the potential to dramatically reduce simulation time. A simulation may require a very small initial timestep as species densities may be unstable early on, but as densities begin to stabilize a larger timestep is permissable.

NOTICE

If you have reason to believe your solution is periodic in nature, be careful when using an adaptive timestep. Convergence may still be achieved, but the timestep may grow to such an extent that the periodic behavior could be hidden or destroyed. It is useful to set a maximum timestep to ensure this does not occur.

Outputs

The [Outputs] block will define the format of the output file. Since *Crane* is traditionally used as a 0D solver, it is recommended to output everything to a CSV file. This will tabulate all Variables and AuxVariables as a function of time, including rate coefficients.

```
[Outputs]
    csv = true
[]
```

Other output options are available on the MOOSE framework website, but for most *Crane* simulations this is sufficient.

One useful output option for larger simulations is suppressing the scalar variable console output. By default, all scalar values in a MOOSE simulation are output in a table to the console while the simulation runs. This is useful for seeing values in real time, but the console quickly becomes cluttered with more than a few variables and reactions. To suppress output, a user must add a console subblock to their [Outputs] block and set the `execute_scalars_on` parameter to 'none'.

```
[Outputs]
    csv = true
    [./console]
        type = Console
        execute_scalars_on = 'none'
    [../]
[]
```

Preconditioning

MOOSE includes a variety of preconditioners that can reduce the number of nonlinear iterations required to achieve convergence. In general, *Crane* simulations only require one type of preconditioning: single matrix preconditioning (`type = SMP`).

```
[Preconditioning]
    [./smp]
        type = SMP
        full = true
    [../]
[]
```

For the vast majority of problems solved with *Crane*, this preconditioner will be sufficient. More information about available preconditioners and advanced options are available on the MOOSE framework website.

2.6 Sample Input File - Lorenz Attractor

This section describes the construction of a sample input file. Rather than use a chemical reaction network, this section will showcase the possibility of using *Crane* on a more general system of PDEs - in this case, the Lorenz attractor.

Problem Statement

The Lorenz attractor is a system of three time-dependent PDEs:

$$\begin{aligned}\frac{\partial X}{\partial t} &= \sigma(Y - X)\\ \frac{\partial Y}{\partial t} &= -XZ + rX - Y\\ \frac{\partial Z}{\partial t} &= XY - bZ\end{aligned}$$

where the factors σ, r, and b are the Prandtl number, the Rayleigh number divided by the critical Rayleigh number, and the geometric factor, respectively. For the classic Lorenz attractor these values are $\sigma = 10$, $r = 28$, and $b = 8/3$. Initial conditions will change the final values of the problem, but the behavior of the attractor will remain the same. For simplicity they may all be set to 1.

Input File and Results

To solve this system of equations in *Crane*, it must first be cast into a system of "reactions". This may be done by recognizing that the terms on the right hand side are essentially source and sink terms. For example, X will "decompose" spontaneously at a rate of σ, providing a sink term for itself. Meanwhile, Y will transform into X at a rate of σ. The system of "reactions" is thus:

$$\begin{aligned} x &\xrightarrow{\sigma} a\\ y &\xrightarrow{\sigma} x + y\\ x &\xrightarrow{r} x + y\\ y &\xrightarrow{1.0} a\\ x + z &\xrightarrow{-1.0} x + y + z\\ x + y &\xrightarrow{1.0} x + y + z\\ z &\xrightarrow{b} a\end{aligned}$$

In this system of reactions, the "product" `a` is intended to be an arbitrary sink term. It is not a nonlinear variable in the system, so it is ignored by *Crane*. With the system of reactions defined, the

Crane input file may be written. There are three scalar nonlinear variables, X, Y, and Z, and there is one `ODETimeDerivative` applied to each variable. In addition, the reaction network may be written with the `ChemicalReactions` block, which will be explained in detail in the following chapter.

```
[Mesh]
  type = GeneratedMesh
  dim = 1
  nx = 1
[]

[Variables]
  [./x]
    family = SCALAR
    order = FIRST
    # initial_condition = 10.1
    initial_condition = 1.0
  [../]

  [./y]
    family = SCALAR
    order = FIRST
    initial_condition = 1.0
  [../]

  [./z]
    family = SCALAR
    order = FIRST
    initial_condition = 1.0
  [../]
[]

[ScalarKernels]
  [./dx_dt]
    type = ODETimeDerivative
    variable = x
  [../]

  [./dy_dt]
    type = ODETimeDerivative
    variable = y
```

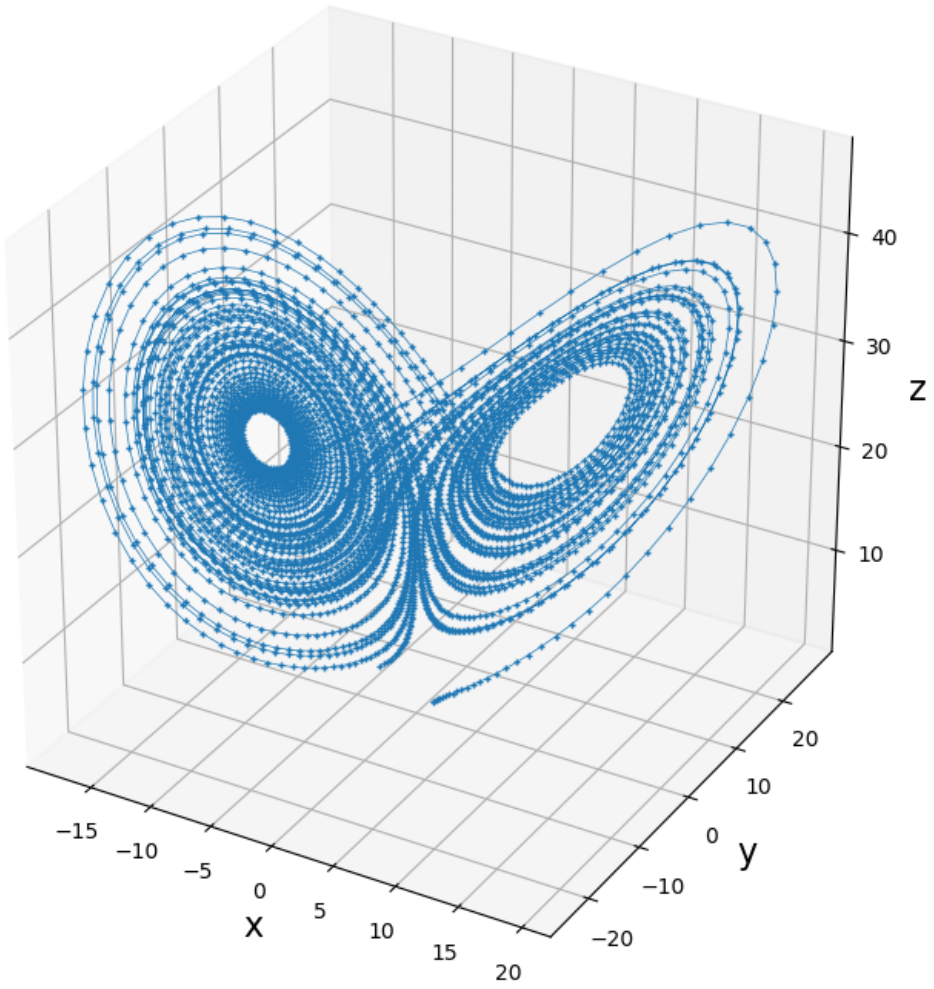
```
  [../]

  [./dz_dt]
    type = ODETimeDerivative
    variable = z
  [../]
[]

[ChemicalReactions]
  [./ScalarNetwork]
    species = 'x y z'
    equation_constants = 'sigma p r'
    equation_values = '10.0 8.0/3.0 28.0'

    reactions = 'x -> a  : {sigma}
                 y -> x + y  : {sigma}
                 x -> x + y  : {r}
                 y -> a  : 1.0
                 x + z -> x + y + z  : -1.0
                 x + y -> x + y + z  : 1.0
                 z -> a  : {p}'
  [../]
[]

[Executioner]
  type = Transient
  end_time = 50
  dt = 0.01
  solve_type = NEWTON
  scheme = crank-nicolson
[]

[Preconditioning]
  [./smp]
    type = SMP
    full = true
  [../]
[]
```

```
[Outputs]
  csv = true
[]
```

In the `ChemicalReactions` block, we have set three equation constants: `sigma`, `p`, and `r`. The values of those constants are set with the `equation_values` parameter. Note that in the list of reactions, those parameters are applied by using curly braces. This indicates to *Crane* that the rate coefficients are "functions". This will be explained in Chapter 3.

The problem is set up to run with $dt = 0.01$ and to run to an end time of 50 "seconds". The Newton solver is utilized, and the numerical scheme is set to Crank-Nicolson. (Note that the Crank-Nicolson scheme is useful for PDEs with chaotic solutions. For a realistic chemical reaction network it is inadvisable to use the Crank-Nicolson solver.) A single matrix preconditioner is defined and a CSV output type is indicated. The output of this program is shown in the figure.

With no additional parameters, the `csv = true` option will automatically create an output file with the string `_out` appended to the input file name. For example, if this input file is named `input_file.i`, the output file will be named `input_file_out.csv` and will be saved in the same directory from which the input file is run.


40
30
z
20
10
20
10
0
y
−10
−20
−15
−10
−5
0
5
10
15
20
x

Chapter 3

Using the ChemicalReactions Block

The `ChemicalReactions` block is a unique input file feature of *Crane* that allows the user to write and solve chemical reactions in a human readable format. Each feature included in the block is detailed here.

Within the block are two available subblocks: **ScalarNetwork** and **Network**. **When using *Crane* as a standalone solver, the user should always use ScalarNetwork.** The only difference between the two is that using the **Network** block no longer utilizes `SCALAR` variables so the problem will exist on the mesh. This use case is generally only used when coupling *Crane* to other MOOSE applications, as *Crane* has no available Kernels that deal with spatial variation (diffusion, advection, etc.). The user is free to use the Network block, but the `SCALAR` family cannot be used in this case, and the problem will take longer to solve when compared to ScalarNetwork since the problem will need to be solved on the mesh.

3.1 Basic Use Case

At its simplest, the **ChemicalReactions** block requires only two parameters as input: (1) the list of active species, and (2) a list of reactions.

```
[ChemicalReactions]
    [./ScalarNetwork]
        species = 'e Ar Ar+'

        reactions = 'e + Ar -> e + e + Ar+    : 1e-12
                     e + Ar+ + Ar -> Ar + Ar : 1e-25'
    [../]
[]
```

The snippet shown here will perform the following actions behind the scenes:

1. Add one source and one sink ScalarKernel for species e

2. Add one source and one sink ScalarKernel for species Ar
3. Add one source and one sink ScalarKernel for species Ar+
4. Add two AuxVariables to store the rate coefficients of each reaction ($1e-12$ and $1e-25$, respectively).
5. Apply the rate coefficients to the appropriate source and sink ScalarKernels

Thus a simple reaction network is defined with relative ease.

1. **Species list**
   - Takes a list of variable names enclosed in single quotation marks.
   - Note that these **must** be the same names as those declared in the [Variables] block.
   - If any species utilized in the reactions list are not declared here, they will be ignored by the problem. This may be leveraged to create arbitrary sink terms (e.g. a reaction such as $Ar \rightarrow W$ where 'W' is not an active species will simply act as a sink term for Ar, and W will not be included as a species.
   - Due to the way source and sink terms are formulated, all reactants (species on the left side of each reaction equation) **must be either Variables or AuxVariables with specified values**.
2. **Reactions list**
   - Each reaction may be written in the form shown above.
   - The colon character separates the reaction from the **reaction rate coefficient**, which should be in physical units (e.g. $m^3 s^{-1}$).
   - Units of the species initial conditions and rate coefficients must match (if species density is listed in cm^{-3}, rate coefficients must also be in $cm^3 s^{-1}$).
   - Note that *Crane* will recognize an entry such as $2Ar$ as a variable named '2Ar', not as 'Ar + Ar'.
   - Each reaction must be separated by a minimum of a return character ('Enter'). Any spacing beyond that is optional and is only shown here to enhance user readability.

3.2 Rate Coefficients

Rate coefficients are separated from the reaction by a colon character. (A single whitespace character is required before and after the colon character; beyond that whitespace is ignored). There are three different types of rate coefficients that are accepted, as shown in the table below.

Rate Coefficient	Format	Definitions
Constant	$1e0$	A single constant value. This will apply an AuxVariable with a constant value to the problem.
Equation	$\{1e-12*exp(Te/10)\}$	An equation enclosed in curly braces ({}). Any non-species variables included in the equation must be declared by an additional parameter ('equation_variables').
Electron energy distribution solver	EEDF	This option suggests that the rate coefficient is either precomputed from an EEDF solver (Bolsig+ or Bolos) or is intended to be computed on-the-fly by Bolsig+.

EEDF Rate Coefficients

Electron-impact rate coefficients (ionization, excitation, etc.) are traditionally computed through convolving an electron energy distribution function with cross section data. *Crane* includes two separate use cases for computing electron-impact rate coefficients: (1) tabulating rate coefficients with an external program, and (2) using cross sections as input and computing rate coefficients on-the-fly with Bolsig+. Note that for the second case the user must download and include Bolsig+ in crane's home directory separately. **Bolsig+ is not distributed with *Crane*.**

In both cases, the user must write the keyword EEDF in place of a constant or equation-based rate coefficient.

Tabulating EEDF Rate Coefficients

Precomputing rate coefficients and tabulating them as a function of either reduced electric field or electron temperature is the fastest approach. Additional options must be input for this use case.

```
[ChemicalReactions]
  [./ScalarNetwork]
    species = 'e Ar* Ar+ Ar Ar2+'
    file_location = 'RateCoefficients'

    reactions = 'e + Ar -> e + e + Ar+          : EEDF
                 e + Ar -> Ar* + e              : EEDF
                 e + Ar* -> Ar + e              : EEDF
```

```
                e + Ar* -> Ar+ + e + e        : EEDF'
   [../]
[]
```

The option 'file_location' denotes a separate file where the tabulated rate coefficients are stored. In this case, *Crane* will look in a directory named `RateCoefficients` for four text files, one for each reaction. Note that the directory location is with respect to the current directory. For example, if the user runs *Crane* from the `crane/problems/` directory, a subdirectory named 'RateCoefficients' should be located in the problems directory: `crane/problems/RateCoefficients/`.

There are two options for naming the tabulated rate coefficient files. By default, *Crane* will search for a file named after the reaction. For example, for the first reaction in the list shown above *Crane* will search in the `RateCoefficients` directory for a file named `reaction_e + Ar -> e + e + Ar+.txt`. A second option is to use any file naming convention the user would like, and simply name the file immediately after the `EEDF` declaration. For example, if the reaction files were named `reaction1.txt`, `reaction2.txt`, etc. the user may specify this in the following way:

```
[ChemicalReactions]
  [./ScalarNetwork]
    species = 'e Ar* Ar+ Ar Ar2+'
    file_location = 'RateCoefficients'

    reactions = 'e + Ar -> e + e + Ar+         : EEDF (reaction1.txt)
                 e + Ar -> Ar* + e             : EEDF (reaction2.txt)
                 e + Ar* -> Ar + e             : EEDF (reaction3.txt)
                 e + Ar* -> Ar+ + e + e        : EEDF (reaction4.txt)'
   [../]
[]
```

In this case *Crane* will search for the specified filenames in the `RateCoefficients` directory.

Gas and Electron Energy Changes

Crane includes the capability of tracking gas and/or electron energy changes due to reactions. If a reaction has an associated energy gain or loss associated with it, this value may be indicated after the rate coefficient enclosed in square brackets. An example is shown below.

```
[ChemicalReactions]
  [./ScalarNetwork]
    species = 'e Ar* Ar+ Ar Ar2+'
    file_location = 'RateCoefficients'
```

```
    electron_energy = 'Te'
    gas_energy = 'Tgas'

    reactions = 'e + Ar -> e + e + Ar+          : EEDF [-15.76]
                 e + Ar -> Ar* + e              : EEDF [-11.6]
                 e + Ar* -> Ar + e              : EEDF [11.6]
                 e + Ar* -> Ar+ + e + e         : EEDF [-4.16]'
  [../]
[]
```

The energy values are in eV and are tabulated with respect to the electron energy, so a value of -15.76 means that the reaction will cause an energy *loss* of 15.76 eV for electrons and an energy *gain* of 15.76 eV for the neutral gas. Note that these values will be ignored if no electron or gas energy is set. Both energy variables do not need to be tracked for the energy changes to be applied. (Electron energy changes can be tracked even if gas energy is not tracked, and vice-versa.)

Energy changes may be applied to *any* reaction and are not isolated to EEDF reactions.

Equation-Based Rate Coefficients

Many reaction rate coefficients in literature are given in terms of electron or gas temperature, often in Arrhenius form (e.g. $k = Aexp(\frac{-E_a}{kT})$). *Crane* allows the user to write rate coefficients as equations, as shown in table 3.2 above. To do so, rather than writing a constant value or `EEDF`, the user simply writes the equation enclosed in brackets. Any Variables and AuxVariables used in these equations must be specified by a parameter named `equation_variables`, and similarly any constants must be specified by `equation_constants` which defines the constants and `equation_values` which defines the values of each constant. An example is shown below.

```
[ChemicalReactions]
  [./ScalarNetwork]
    species = 'e Ar* Ar+ Ar Ar2+'

    equation_constants = 'Tgas J pi'
    equation_values = '300 2.405 3.141'
    equation_variables = 'Te'


    reactions = 'Ar2+ + e -> Ar* + Ar           : {8.5e-7*((Te/1.5)*11600/300.0)^(-0.67)}
                 Ar2+ + Ar -> Ar+ + Ar + Ar     : {(6.06e-6/Tgas)*exp(-15130.0/Tgas)}
                 Ar* + Ar* -> Ar2+ + e          : 6.0e-10
                 Ar+ + e + e -> Ar + e          : {8.75e-27*((Te/1.5)^(-4.5))}
                 Ar* + Ar + Ar -> Ar + Ar + Ar  : 1.399e-32
                 Ar+ + Ar + Ar -> Ar2+ + Ar     : {2.25e-31*(Tgas/300.0)^(-0.4)}'
  [../]
```

```
[]
```

A mix of equation-based and constant rate coefficients are shown in the input file snippet. Three equation constants are defined: Tgas, J, and pi. Their values are assigned as 300, 2.405, and 3.141, respectively. In addition, an equation variable Te is defined. Note that Te does not have an assigned value. Since it is declared as an equation variable, it must be either a nonlinear variable or an auxiliary variable in the system. In this case since it represents electron temperature and not a species density, Te is a scalar AuxVariable.

3.3 Input Options

The `ChemicalReactions` block comes equipped with many input options that may be applied. The full list of available options is detailed in this section. Note that a `vector, string` or `vector, Variable Name` type means that the user may input multiple entries inside single quotes. For example, the species input parameter is a `vector, Variable Name` type and is declared in the following way: `species = 'e Ar Ar+'`.

Bold parameters are required inputs. The rest are optional.

Parameter	Type	Definition
use_bolsig	true / false	Whether or not to use Bolsig+ dynamically. Default: false.
species	vector, Variable Name	The list of active species. These must all be either nonlinear variables or AuxVariables.
aux_species	vector, string	Any value in the species list that is an AuxVariable instead of a Variable.
reactions	vector, string	The list of reactions. Each reaction is separated by the return/enter key and must have at least (1) the reaction equation and (2) the rate coefficient. (1) and (2) must be separated by a colon character.
file_location	string	The name of the directory that is storing the EEDF rate coefficients. Required only if EEDF rate coefficient type is being used.
use_log	true / false	Indicates if the species densities are in log form. Default: false.
electron_density	Variable Name	The name of the variable which represents electron density. Required only if (a) use_bolsig = true or (b) electron_energy and/or gas_energy variables are set.
electron_energy	Variable Name	The name of the variable/auxiliary variable representing electron energy. If this factor is set, energy changes due to reactions are taken into account (if available).
gas_energy	Variable Name	The name of the variable/auxiliary variable representing electron energy. If this factor is set, gas temperature changes due to reactions are taken into account (if available).
sampling_variable	string	The name of the variable/auxiliary variable whose value is being used to sample from rate coefficient data files. Typically electron temperature or reduced electric field. (Required only if EEDF reactions are set.)
equation_constants	vector, string	The names of any constants that appear in any of the equation-based rate coefficients.
equation_values	vector, string	The values associated with each of the equation_constants that are set.
equation_variables	vector, Variable Name	The names of any variables (both nonlinear and auxiliary) that appear in equation-based rate coefficients.

Input Options - Bolsig+

If `use_bolsig` is set to true, additional parameters must be set in order to properly set up the dynamic use of Bolsig+.

NOTICE

Bolsig+ is NOT distributed with *Crane*! In order to apply the `use_bolsig` option, the `bolsigminus` version of Bolsig+ must be downloaded and moved into the `crane/` directory (in the same location as the crane executable).
Note that this feature is still in its beta phase and sees frequent updates as of February 2019. It is recommended that current users precompute rate coefficients for production runs.

Credits and Copyright Statement

Crane was developed at the Laboratory for Computational Plasma Physics at the University of Illinois at Urbana-Champaign.

- Shane Keniley, Graduate Research Assistant
- Dr. Davide Curreli, Assistant Professor

Crane is protected under the GNU Lesser General Public License and is freely available for download. The product is an ongoing project and is subject to frequent updates.