Saint Petersburg
State University
www.spbu.ru

20th International Conference on the Methods of Aerophysical Research
(ICMAR 2020)
November 01 - 07, 2020 Novosibirsk, Russia

Machine Learning for State-to-State

L. Campoli, M. Bushmakova, V. Gorikhovskii,
O. Kunova, E. Kustova, M. Melnik

November 3, 2020

Outline

Saint Petersburg
State University
www.spbu.ru

Introduction

Target problem: multi-component reacting gas mixture flows under strong vibrational and chemical non-equilibrium conditions.

- Most detailed model of physical gas dynamics taking into account state-to-state vibrational and chemical kinetics [1];
- Equations for macroscopic parameters of the flow are coupled to the equations of physical-chemical kinetics;
- Production terms describe the variation of vibrational level populations due to the non-equilibrium kinetic processes;
- Transport coefficients, heat fluxes, diffusion velocities directly depend on non-equilibrium distributions;
- Computationally demanding approach.

Introduction

The master eqs. for the vibrational and chemical relaxation behind a shock wave include the 1D conservation eqs. of momentum and total energy coupled with the eqs. for the vibrational state populations n_{ci} of molecular species and for the number densities n_c of atomic species [2]:

$$\rho v \frac{\partial v}{\partial x} + \frac{\partial p}{\partial x} = 0, \tag{1}$$

$$v \frac{\partial E}{\partial x} + (E + p) \frac{\partial v}{\partial x} = 0, \tag{2}$$

$$v \frac{\partial n_{ci}}{\partial x} + n_{ci} \frac{\partial v}{\partial x} = R_{ci}^{vibr} + R_{ci}^{react}, \quad i = 0, \dots l_c, \quad c = 1, \dots l_m, \tag{3}$$

$$v \frac{\partial n_c}{\partial x} + n_c \frac{\partial v}{\partial x} = R_c^{react}, \quad c = 1, \dots l_a. \tag{4}$$

The rates of the variation of the vibrational level population as result of VT, VV energy transitions and chemical reactions are:

$$R_{ci} = R_{ci}^{vibr} + R_{ci}^{react} = R_{ci}^{VT} + R_{ci}^{VV} + R_{ci}^{2\rightleftharpoons 2} + R_{ci}^{2\rightleftharpoons 3} \tag{5}$$

$$R_{ci}^{VT} = \sum_{d=a,m} n_d \sum_{i' \neq i} \left(n_{ci'} k_{c,i'i}^{d} - n_{ci} k_{c,ii'}^{d} \right) \tag{6}$$

$$R_{ci}^{VV} = \sum_{dki'k'} \left(n_{ci'} n_{dk'} k_{c,i'i}^{d,k'k} - n_{ci} n_{dk} k_{c,ii'}^{d,kk'} \right) \tag{7}$$

$$R_{ci}^{2\rightleftharpoons 2} = \sum_{dc'd'} \sum_{ki'k'} \left(n_{c'i'} n_{d'k'} k_{c'i',ci}^{d'k',dk} - n_{ci} n_{dk} k_{ci,c'i'}^{dk,d'k'} \right) \tag{8}$$

$$R_{ci}^{2\rightleftharpoons 3} = \sum_{dk} n_{dk} \left(n_{c'} n_{f'} k_{rec,ci}^{dk} - n_{ci} k_{ci,diss}^{dk} \right) \tag{9}$$

Introduction

Three main sub-tasks have been identified:

- Regression of relaxation source terms
- Machine learning and ODE solver coupling
- Neural network for post-shock relaxation

Regression of relaxation source terms

Saint Petersburg State University
www.spbu.ru

- Supervised machine learning task
- Multi-input/output regression
- Dataset:

$$\begin{cases} input: & n_{ci}, n_c, v, T, x_s \\ output: & R_{ci} \end{cases}$$

- `train_test_split`: 75/25
- `StandardScaler` data standardization
- Hyper-parameters tuning by `GridSearchCV`
- Random seeding for reproducibility

Regression of relaxation source terms

Saint Petersburg State University
www.spbu.ru

Table: Comparison of several MLAs for regression of relaxation terms

Regression Method	MAE	MSE	RMSE	T_{train} [s]	$T_{predict}$ [s]
Kernel Ridge	7.868e-08	3.800e-14	1.949e-07	7.612	0.075
Support Vector Machine	1.236e-02	2.109e-04	1.452e-02	5.317	0.008
k-Nearest Neighbour	8.655e-04	2.659e-06	1.630e-03	0.002	0.004
Gaussian Process	7.235e-07	2.436e-12	1.561e-06	118.391	0.098
Decision Tree	2.417e-03	1.623e-05	4.028e-03	0.003	0.0003
Random Forest	1.140e-03	5.016e-06	2.239e-03	4.362	0.038
Extra Trees	1.595e-03	6.005e-06	2.450e-03	2.279	0.202
Gradient Boosting	2.300e-03	1.478e-05	3.844e-03	4.829	0.006
Hist Gradient Boosting	6.098e-03	1.395e-04	1.181e-02	14.385	0.042
Multi-layer Perceptron	6.023e-03	7.539e-05	8.682e-03	11.322	0.009

$T_{Matlab}^{N_2/N} = 0.003\ s$

$T_{Matlab}^{Air5} = 0.7\ s$

Machine learning and ODE solver coupling

- MATLAB code: ODE solver + source term function
- Pre-trained "best" performing ML algorithm
- Directly call Python from MATLAB

- Where to apply the ML?

1. regression of chemical reaction rate coefficients, k_{ci} k_{ci}
2. regression of chemical reaction relaxation terms, R_{ci}, B
3. regression of the r.h.s inside ODE function call, dy
4. regression of the ODE solver function call output, $[X, Y]$ XY

Listing 1: Source terms calculation function

```
function dy = rhs(t,y)

% A*X=B
A = zeros(sum(l)+4,sum(l)+4);

% computation of rate coefficients, k_{ci} ...

% computation of relaxation terms, R_{ci} ...

B = zeros(lall+4,1);
B(1:l1)          = RDn2 + RZn2 + RVTn2 + RVVn2 + RVVsn2;
B(l1+1:l1+l2)    = RDo2 + RZo2 + RVTo2 + RVVo2 + RVVso2;
B(l1+l2+1:lall) = RDno + RZno + RVTno + RVVno + RVVsno;
B(lall+1) = -sum(RDno)-2*sum(RDn2)-sum(RZn2)+sum(RZo2);
B(lall+2) = -sum(RDno)-2*sum(RDo2)+sum(RZn2)-sum(RZo2);

dy = A^{-1}*B;
```

Machine learning and ODE solver coupling

- ML call performed instead of the ODE solver

Table: Comparison of time-to-simulation for MATLAB and ML solutions for the same number of integration points.

	N_2/N		Air5	
	MATLAB	ML	MATLAB	ML
Time [s]	7.3541	6.8475	1874.7	6.3974

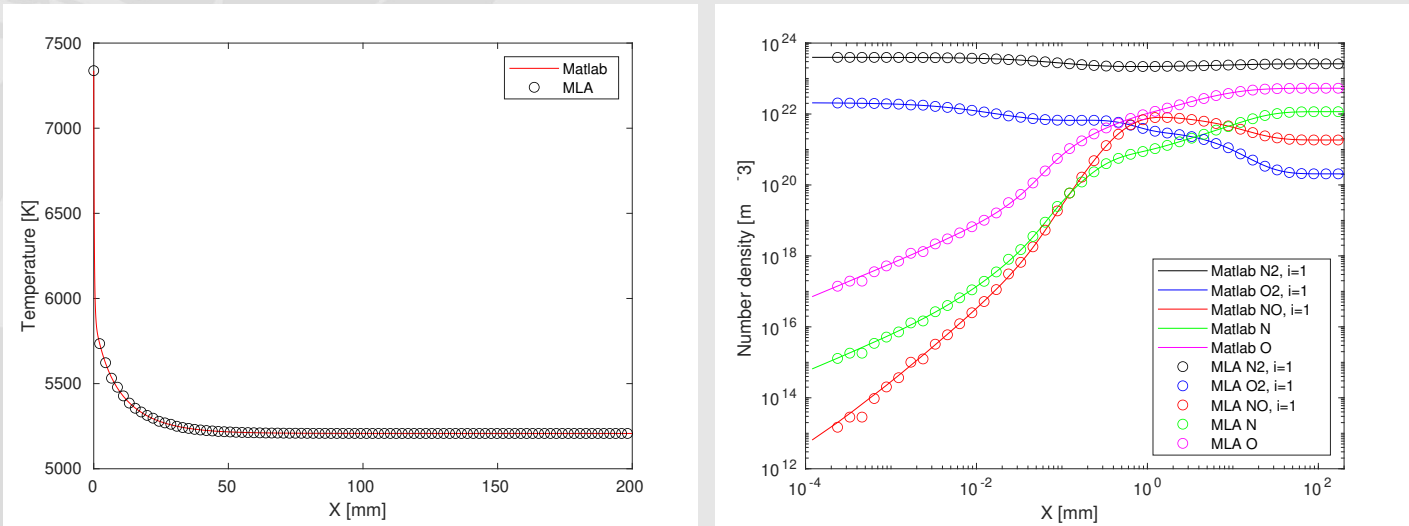


Figure: Comparison of MATLAB and ML solution for the 1D reacting shock flow for air5 in STS approach.

Machine learning and ODE solver coupling

- The ML call is performed within the ODE system integration before the matrix inversion

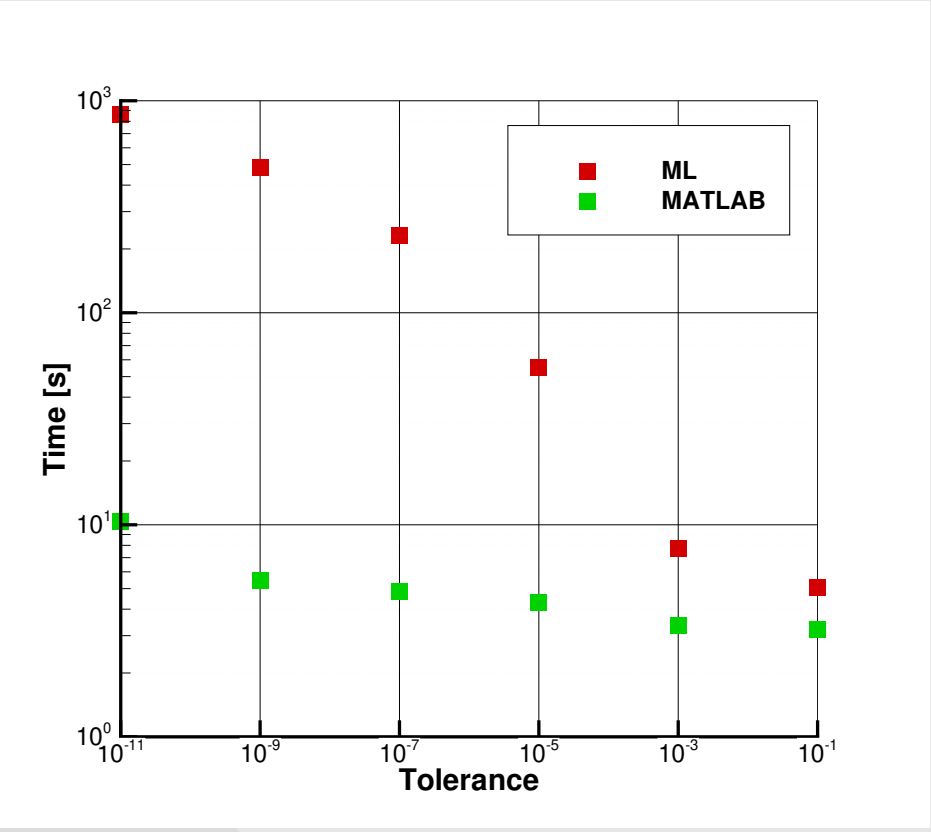


Figure: Comparison of MATLAB and ML time-to-solution for the 1D reacting shock flow for N_2/N in STS approach.

Machine learning and ODE solver coupling

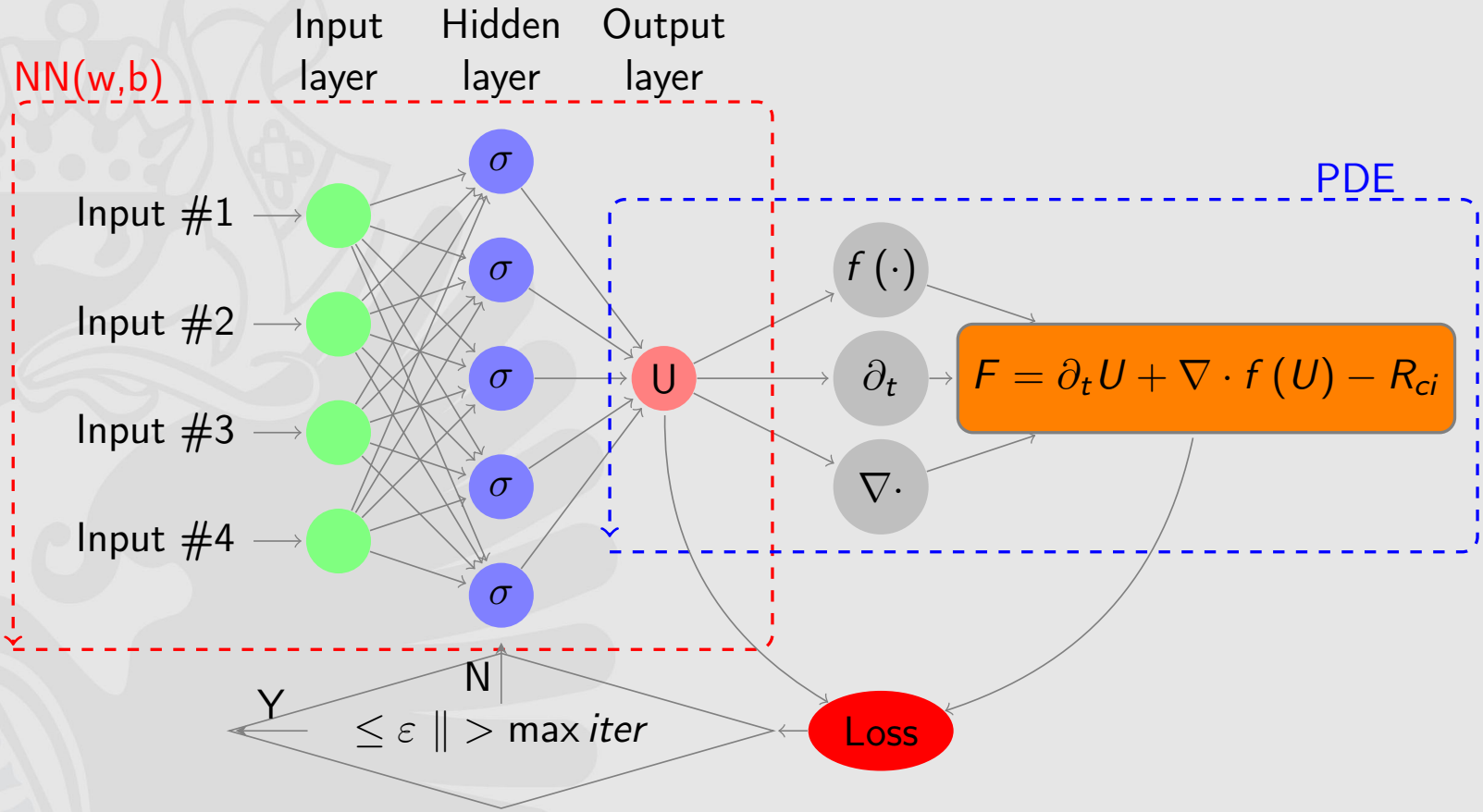
The problem with using ML to predict the integration of the relaxation terms is that the accuracy of the predicted values ($\sim 10e-3 \div 10e-5$) is not sufficient when such values are repeatedly fed into the ODE solver.

- ML based predictions can be quicker
- ML methods can have a profound impact on the workload when hybridized with traditional CFD algorithms
- ML for IVP is exceptionally difficult if used on primary state variables because it is difficult to correct problems, but ML is suitable for secondary property prediction
- For BVP is much easier to hybridize

$\rightarrow$This issue may be solved with Neural ODE [3] ...

Neural network for post-shock relaxation

Saint Petersburg State University
www.spbu.ru

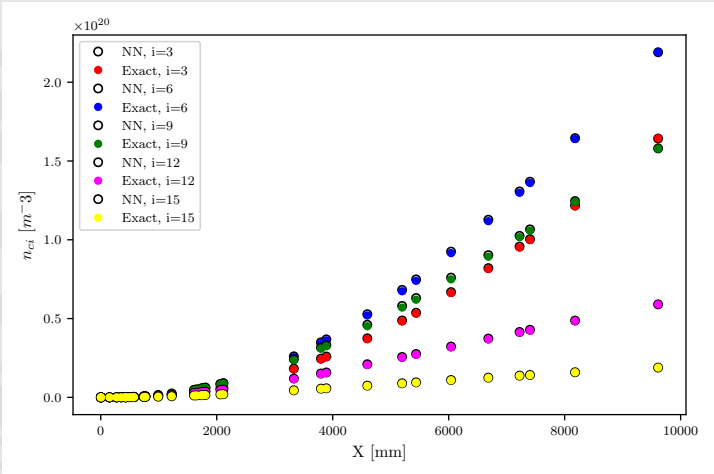


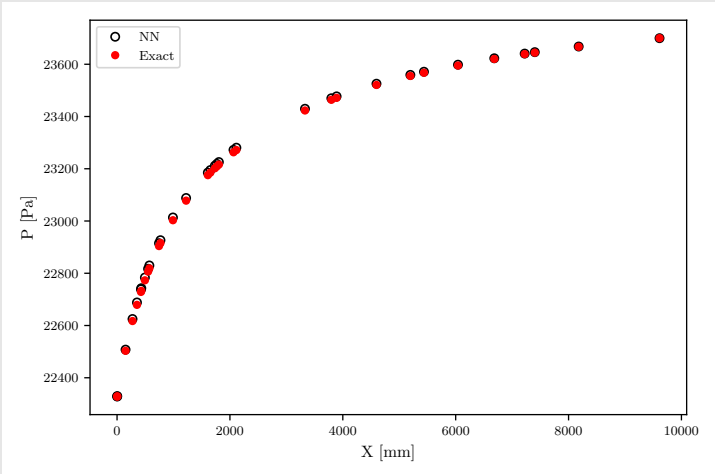
Schematic of PINN [4, 5] for the Euler equations for STS.

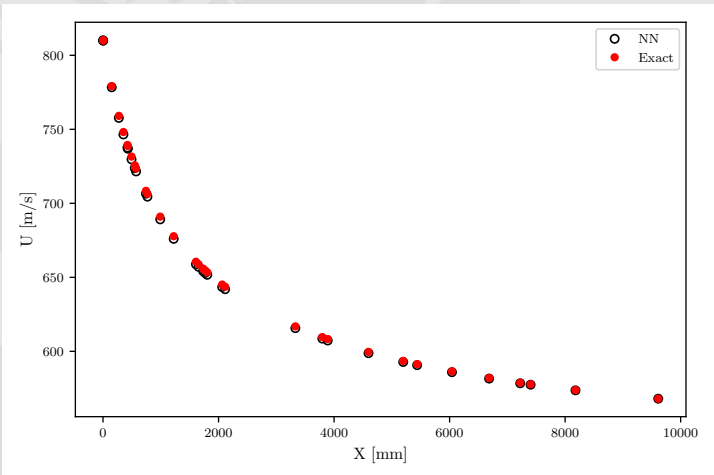
layers = [1, 15, 25, 25, 15, 100] for N_2/N binary mixture

Neural network for post-shock relaxation

Saint Petersburg State University
www.spbu.ru







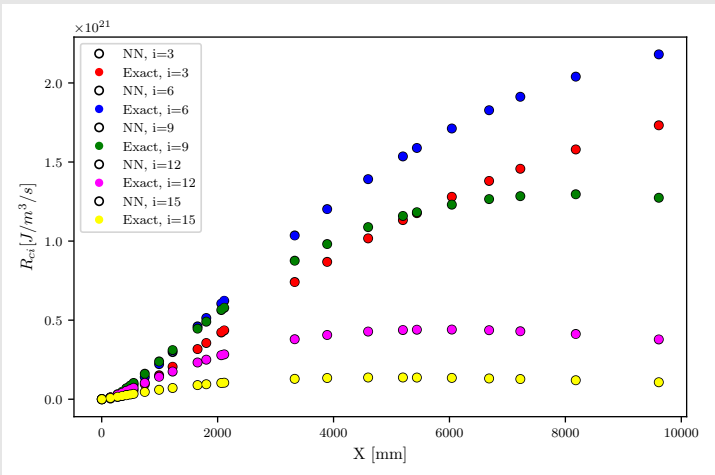


Figure: Solution of the STS 1D shock flow relaxation computed by PINN.

Conclusions

By applying ML methods to STS 1D shock flow we noted that:

- MLAs are very quick but less accurate then traditional methods
- integration with ODE/PDE solver should be done with care and it could allow a significant speed-up
- neural ODE could be an interesting option (i.e. Julia [6])
- NN/PINNs appear to be a viable technology also for STS

Fostered by these preliminary results, we plan to:

- extend the regression analysis to transport coefficients
- integrate MLAs (kinetics+transport) into a CFD solver [7]

https://github.com/lkampoli/ML4STS.git

Bibliography I

E.A. Nagnibeda and E.V. Kustova.

Nonequilibrium Reacting Gas Flows. Kinetic Theory of Transport and Relaxation Processes.

Springer-Verlag, Berlin, Heidelberg, 2009.

L Campoli, O Kunova, E Kustova, and M Melnik.

Models validation and code profiling in state-to-state simulations of shock heated air flows.

Acta Astronautica, 2020.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud.

Neural ordinary differential equations.

In Advances in neural information processing systems, pages 6571–6583, 2018.

Bibliography II

Zhiping Mao, Ameya D Jagtap, and George Em Karniadakis.

Physics-informed neural networks for high-speed flows.

Computer Methods in Applied Mechanics and Engineering, 360:112789, 2020.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis.

Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.

Journal of Computational Physics, 378:686–707, 2019.

Chris Rackauckas, Mike Innes, Yingbo Ma, Jesse Bettencourt, Lyndon White, and Vaibhav Dixit.

Diffeqflux. jl-a julia library for neural differential equations.

arXiv preprint arXiv:1902.02376, 2019.

Bruno Lopez and Mario Lino Da Silva.

Spark: a software package for aerodynamics, radiation and kinetics.

In 46th AIAA thermophysics conference, page 4025, 2016.

Bibliography III

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al.

Scikit-learn: Machine learning in python.

the Journal of machine Learning research, 12:2825–2830, 2011.

Acknowledgement

- Russian Science Foundation, grant 19-11-00041

APPENDIX: source terms (1)

Source terms R_{ci}: variation of vibrational level population and atomic number density by vibrational energy exchanges and chemical reactions

$$R_{ci} = R_{ci}^{vibr} + R_{ci}^{react} = R_{ci}^{VT} + R_{ci}^{VV} + R_{ci}^{2\rightleftharpoons 2} + R_{ci}^{2\rightleftharpoons 3}$$

$$R_{ci}^{VT} = \sum_{d=a,m} n_d \sum_{i' \neq i} \left(n_{ci'} k_{c,i'i}^{d} - n_{ci} k_{c,ii'}^{d} \right)$$

$$R_{ci}^{VV} = \sum_{dki'k'} \left(n_{ci'} n_{dk'} k_{c,i'i}^{d,k'k} - n_{ci} n_{dk} k_{c,ii'}^{d,kk'} \right)$$

$$R_{ci}^{2\rightleftharpoons 2} = \sum_{dc'd'} \sum_{ki'k'} \left(n_{c'i'} n_{d'k'} k_{c'i',ci}^{d'k',dk} - n_{ci} n_{dk} k_{ci,c'i'}^{dk,d'k'} \right)$$

$$R_{ci}^{2\rightleftharpoons 3} = \sum_{dk} n_{dk} \left(n_{c'} n_{f'} k_{rec,ci}^{dk} - n_{ci} k_{ci,diss}^{dk} \right)$$

APPENDIX: source terms (2)

The following processes are included to the kinetic scheme: VV and VV' vibrational energy exchanges with the same and different chemical species; single-quantum VT vibration-translation energy exchanges; all kinds of state-resolved dissociation reactions; Zeldovich exchange reactions taking into account vibrational excitation of both reagents and products.

$$\mathrm{VT}: AB(i) + M \rightleftarrows AB(i \pm 1; i \pm 2) + M, \tag{10}$$

$$\mathrm{VV}: AB(i) + AB(f) \rightleftarrows AB(i \pm 1) + AB(f \mp 1), \tag{11}$$

$$\mathrm{VV'}: AB(i) + CD(f) \rightleftarrows AB(i \pm 1) + CD(f \mp 1), \tag{12}$$

$$\mathrm{DR}: AB(i) + M \rightleftarrows A + B + M, \tag{13}$$

$$AB,\ CD = \mathrm{N_2},\ \mathrm{O_2},\ \mathrm{NO}, \quad M = \mathrm{N_2},\ \mathrm{O_2},\ \mathrm{NO},\ \mathrm{N},\ \mathrm{O},$$

$$\mathrm{ER_1}: \mathrm{N_2}(i) + \mathrm{O} \rightleftarrows \mathrm{NO}(f) + \mathrm{N}, \tag{14}$$

$$\mathrm{ER_2}: \mathrm{O_2}(i) + \mathrm{N} \rightleftarrows \mathrm{NO}(f) + \mathrm{O}. \tag{15}$$

Molecular vibrational energy levels are calculated according to the anharmonic oscillator models. The total numbers of excited states are 122 and include 47 states of $\mathrm{N_2}$, 36 of $\mathrm{O_2}$, and 39 of NO.

APPENDIX: source terms (3)

◂ k_{ci}

Energy exchange: $A_{ci} + A_{dk} \rightleftharpoons A_{ci'} + A_{dk'}$

$$k^{d,k'k(0)}_{c,i'i} = k^{d,kk'(0)}_{c,ii'} \frac{s^c_i s^d_k}{s^{c'}_{i'} s^{d'}_{k'}} \frac{Z^{rot}_{ci} Z^{rot}_{dk}}{Z^{rot}_{ci'} Z^{rot}_{dk'}} \exp\left(\frac{\varepsilon^c_{i'} + \varepsilon^d_{k'} - \varepsilon^c_i - \varepsilon^d_k}{kT}\right)$$

Exchange chemical reactions: $A_{ci} + A_{dk} \rightleftharpoons A_{c'i'} + A_{d'k'}$

$$k^{d'k',dk(\mathbf{0})}_{c'i',ci} = k^{dk,d'k'(\mathbf{0})}_{ci,c'i'} \frac{s^c_i s^d_k}{s^{c'}_{i'} s^{d'}_{k'}} \left(\frac{m_c m_d}{m_{c'} m_{f'}}\right)^{\frac{3}{2}} \frac{Z^{rot}_{ci} Z^{rot}_{dk}}{Z^{rot}_{ci'} Z^{rot}_{dk'}} \exp\left(\frac{\varepsilon^{c'}_{i'} + \varepsilon^{d'}_{k'} - \varepsilon^c_i - \varepsilon^d_k}{kT}\right) \exp\left(\frac{D_c + D_d - D_{c'} - D_{d'}}{kT}\right)$$

Dissociation-recombination reactions: $A_{ci} + A_{dk} \rightleftharpoons A_{c'} + A_{f'} + A_{dk}$

$$k^{d(0)}_{rec,ci} = k^{d(0)}_{ci,diss} s^c_i \left(\frac{m_{c'} + m_{f'}}{m_{c'} m_{f'}}\right)^{\frac{3}{2}} h^3 (2\pi kT)^{-\frac{3}{2}} Z^{rot}_{ci} \exp\left(-\frac{\varepsilon^c_i - D_c}{kT}\right)$$

s^c_i is the vibrational statistical weight, D_c and D_d represent the dissociation energy of the molecule c and d, respectively, Z^{rot}_{ci} is the rotational partition function and the prime denotes the energy levels of particles after a collision.

APPENDIX: NN/PINN (0)

Loss of the NN (mean squared error, MSE):

$$NN_{Loss} = MSE = \frac{1}{N_s}\sum_{k=1}^{N_s}(y_i^{pred} - y_i^{truth})^2$$

Loss of the PINN:

$$PINN_{Loss} = MSE + RES = \frac{1}{N_s}\sum_{k=1}^{N_o}\sum_{i=1}^{N_s}(y_{ik}^{pred} - y_{ik}^{truth})^2 + \frac{1}{N_s}\sum_{j=1}^{N_e}\sum_{i=1}^{N_s}(e_{ij}^2)$$

y is the output variable, N_s is the number of datapoints in the dataset, N_o and N_e the number of output variables and governing equations, respectively, and e_{ij} the errors of the "physics":

$$e_1 = \rho v\frac{\partial v}{\partial x} + \frac{\partial p}{\partial x} = \rho v * tf.gradients(v, x) + tf.gradients(p, x)$$

APPENDIX: NN/PINN (1)

Listing 2: Neural Network (NN)

```
def neural_net(self, X, weights, biases):
    num_layers = len(weights) + 1
    H = 2.0*(X - self.lb)/(self.ub - self.lb) - 1.0
    for l in range(0,num_layers-2):
        W = weights[l]
        b = biases[l]
        H = tf.tanh(tf.add(tf.matmul(H, W), b))
    W = weights[-1]
    b = biases[-1]
    Y = tf.add(tf.matmul(H, W), b)
    return Y
```

APPENDIX: NN/PINN (2)

Listing 3: Physics Informed Neural Network (PINN)

```
def PDE(self, x):
    # NN
    NN = self.neural_net(tf.concat([x], 1),
                         self.weights, self.biases)
    # Physics Informed NN
    nci_u_x  = tf.gradients(n1*u, x)[0]
    mass_flow_grad = tf.gradients(rho*u, x)[0]
    momentum_grad = tf.gradients((rho*u*u + p), x)[0]
    energy_grad = tf.gradients((rho*E + p)*u, x)[0]
    gamma = 1.4
    state_res = p - rho*(gamma-1.0)*(E-0.5*gamma*u*u)
    eqni[:] = nci_u_x - Rci[:]
    return nci[:], rho, u, p, E, Rci[:],
           mass_flow_grad, momentum_grad, energy_grad,
           state_res, eqni[:]
```

ML and ODE solver coupling (1)

◂ XY

Listing 4: Stiff ODE solver unction call

```
options = odeset('RelTol', 1e-12, 'AbsTol', 1e-12);
[X,Y] = ode15s(@rhs, xspan, Y0, options);
```

Listing 5: Scikit-learn [8] regressor function call

```
for i = 1:dx:npoints
    input = xspan(i);
    RHS = py.run_regression.regressor(input);
    RHS = double(RHS);
end
```

APPENDIX: ML and ODE solver coupling (2)

Listing 6: MLA regressor

```
import numpy as np
import joblib

def regressor(input):
    # Load scalers
    sc_x = load(open('scaler_x.pkl', 'rb'))
    sc_y = load(open('scaler_y.pkl', 'rb'))
    # Load model
    regr = load('model.sav')
    # Build array of inputs for prediction
    input = np.asarray(input).reshape(1,-1)
    # Scaler input arguments
    input = sc_x.transform(input)
    # Prediction
    y_regr = regr.predict(input)
    # Inverse transformation
    y_regr = sc_y.inverse_transform(y_regr)
    return y_regr
```