SUNSET-code documentation

J. R. C. King*
School of Engineering, The University of Manchester, UK
(Dated: July 20, 2024)

This document details the governing equations and numerical methods used in the **S**calable **U**nstructured **N**ode-**SET** code (aka. SUNSET code), and provides brief instructions for configuration and running of the code.

I. INTRODUCTION

This document describes the **S**calable **U**nstructured **N**ode-**SET** code (aka. SUNSET code). The code is designed for direct numerical simulations of compressible turbulent reacting flows in non-trivial geometries. It builds on the work in [1, 2], using the Local Anisotropic Basis Function Method (LABFM) for spatial discretisation, combined with high-order finite differences in the third dimension, and a third-order Runge-Kutta time integration. This guide should be read in conjunction with the preprint [3].

A. Key features at a glance:

- Solves the fully compressible multi-species reacting Navier-Stokes equations, with detailed chemistry, on an unstructured node-set.
- Resolution may be non-uniform in space, including along boundaries.
- Spatial discretisation uses LABFM in the $x_1 - x_2$ plane, combined with high-order finite differences in the (homogeneous) third dimension x_3.
- High-order (up to 10^{th}) discretisation in space dropping to 4^{th} order at boundaries.
- 3^{rd} order explicit Rungke-Kutta time integration with adaptive error control.
- Boundary conditions using NSCBC: (non-reflecting) inflows & outflows, isothermal or adiabatic walls.
- Periodic/symmetry/low-order-wall boundaries also available.
- Written in `Fortran 90`, accelerated with MPI & OpenMP.

B. A comment on notation

Subscripts i, j, and k are vector indices for Einstein notation, and for these indices, repetition implies summation. Subscripts α and β indicate chemical species. Subscripts a and b indicate collocation point indices. Subscripts l and L are used for polynomial coefficient indices. Subscript m is for chemical reaction number (regular-size m is polynomial order). Note that this is consistent with [3], but not [1, 2], in which we used i and j to indicate collocation point indices.

* jack.king@manchester.ac.uk

II. THE GOVERNING EQUATIONS

The code solves the compressible Navier-Stokes equations in conservative form

$$\frac{\partial \rho}{\partial t} + \frac{\partial \rho u_k}{\partial x_k} = 0 \tag{1a}$$

$$\frac{\partial \rho u_i}{\partial t} + \frac{\partial \rho u_i u_k}{\partial x_k} = -\frac{\partial p}{\partial x_i} + \frac{\partial \tau_{ki}}{\partial x_k} + \rho f_i \tag{1b}$$

$$\frac{\partial \rho E}{\partial t} + \frac{\partial \rho u_k E}{\partial x_k} = -\frac{\partial p u_k}{\partial x_k} - \frac{\partial q_k}{\partial x_k} + \frac{\partial \tau_{ki} u_i}{\partial x_k} + \rho \sum_\alpha Y_\alpha f_k \left(u_k - V_{\alpha,k}\right) \tag{1c}$$

$$\frac{\partial \rho Y_\alpha}{\partial t} + \frac{\partial \rho u_k Y_\alpha}{\partial x_k} = \dot{\omega}_\alpha - \frac{\partial \rho V_{\alpha,k} Y_\alpha}{\partial x_k} \qquad \forall \alpha \in [1, N_S] \tag{1d}$$

where ρ is the density, u_i the i-th component velocity, p the pressure, E the total energy, τ the viscous stress, f a body force, q the heat flux, Y_α the concentration of species $\alpha \in [1, N_S]$, and $V_{\alpha,k}$ is the k-th component of the molecular diffusion velocity of species α. $\dot{\omega}_\alpha$ is the production rate of species α.

The viscous stress is defined as

$$\tau_{ki} = \mu \left(\frac{\partial u_k}{\partial x_i} + \frac{\partial u_i}{\partial x_k} - \frac{2}{3} \frac{\partial u_j}{\partial x_j} \delta_{ik} \right), \tag{2}$$

where μ is the dynamic viscosity and δ_{ik} is the Kronecker delta. The energy is

$$E = \sum_\alpha h_\alpha Y_\alpha - \frac{p}{\rho} + \frac{1}{2} u_i u_i \tag{3}$$

where the sum over α is over all $\alpha \in [1, N_S]$, and h_α is the enthalpy of species α. The molecular diffusion term in (1d) is modelled as a Fickian diffusion process with

$$-\rho V_{\alpha,k} Y_\alpha = \rho \hat{V}_{\alpha,k} Y_\alpha - Y_\alpha \sum_{\beta \in [1, N_S]} \rho \hat{V}_{\beta,k} Y_\beta, \tag{4}$$

in which $\rho \hat{V}_{\alpha,k} Y_\alpha$ is obtained from the diffusion model (either constant Lewis numbers or mixtured-averaged), and the final term is a correction to ensure that

$$\sum_\alpha Y_\alpha = 1. \tag{5}$$

The heat flux vector is then given by

$$q_k = -\lambda \frac{\partial T}{\partial x_k} + \sum_\alpha \rho V_{\alpha,k} Y_\alpha h_\alpha, \tag{6}$$

where λ is the thermal conductivity of the mixture. The system is closed with an ideal gas equation of state

$$\frac{p}{\rho} = R_{mix} T = \frac{R_0 T}{W_{mix}} = R_0 T \sum_\alpha \frac{Y_\alpha}{W_\alpha} \tag{7}$$

where R_{mix} is the gas constant for the mixture, R_0 is the universal gas constant, and W_α is the molar mass of species α, and W_{mix} is the molar mass of the mixture.

A. Evaluation of temperature and heat capacity

The temperature dependence of the specific heat capacity of species α is given by a polynomial of order L

$$c_{p,\alpha} = \sum_{l=1}^{L} a_{\alpha,l} T^{l-1}, \tag{8}$$

which is valid over a specified temperature range $T \in [T_{low}, T_{high}]$, and in which the coefficients $a_{\alpha,l}$ are constructed to fit the standard NASA polynomials [4]. Integration of this yields an expression for the enthalpy

$$h_\alpha = \sum_{l=1}^{L} \frac{a_{\alpha,l}}{l} T^l + a_{\alpha,L+1}, \tag{9}$$

whilst the temperature derivative of $c_{p,\alpha}$ is

$$\frac{dc_{p,\alpha}}{dT} = \sum_{l=1}^{L} (l-1)\, a_{\alpha,l} T^{l-2}. \tag{10}$$

Noting that the specific heat capacity of species α is R_0/W_α, where R_0 is the universal gas constant and W_α is the molar mass of species α, the energy can be written as

$$E = \sum_\alpha Y_\alpha \left(h_\alpha - R_0 \frac{T}{W_\alpha} \right) + \frac{1}{2} \boldsymbol{u} \cdot \boldsymbol{u}. \tag{11}$$

Substituting the polynomial for h_α and collecting terms in powers of T, we can write

$$f(T) = C_0 + \sum_{l=1}^{L} C_l T^l = 0, \tag{12}$$

where the coefficients C are given by

$$C_0 = \frac{1}{2} \boldsymbol{u} \cdot \boldsymbol{u} - E + \sum_\alpha Y_\alpha a_{\alpha,L+1} \tag{13a}$$

$$C_1 = \sum_\alpha Y_\alpha a_{\alpha,1} - R_{mix} \tag{13b}$$

$$C_l = \sum_\alpha Y_\alpha \frac{a_{\alpha,l}}{l} \qquad \forall l \in [2, L]\,. \tag{13c}$$

The derivative of f is

$$f'(T) = \sum_{l=1}^{L} l C_l T^{l-1}. \tag{14}$$

Having calculated the coefficients C, we solve the nonlinear equation (12) numerically via a Newton-Raphson method to obtain the temperature T.

Note that in the case where $a_{\alpha,1} \neq 0$ and $a_{\alpha,l\in[2,L]} = 0$ (i.e. $c_{p,\alpha}$ is independent of temperature), $f(T)$ is linear in T, and the Newton-Raphson method will converge (exactly) after a single iteration.

B. Transport properties

Two models are implemented for transport properties. The constant Lewis number approximation, and a mixture averaged transport model.

1. Constant Lewis numbers

In this model, the diffusive flux is given by

$$\rho \hat{V}_{\alpha,k} Y_\alpha = -\rho D_\alpha \frac{\partial Y_\alpha}{\partial x_k}, \tag{15}$$

in which D_α is the molecular diffusivity of species α. The temperature dependence of transport properties is modelled by the relationship

$$\mu = \mu_{ref} \left(\frac{T}{T_{ref}} \right)^{r_T}, \tag{16}$$

where μ_{ref} is the viscosity at the reference temperature T_{ref}, and $r_T = 0.7$ is a constant exponent. The mixture thermal conductivity is given by

$$\lambda = \frac{\mu c_p}{Pr}, \tag{17}$$

in which c_p is the heat capacity of the mixture, and Pr is the Prandtl number, taken as constant. The molecular diffusivity of species α is given by

$$D_\alpha = \frac{\lambda}{\rho c_p Le_\alpha}, \tag{18}$$

where Le_α is the Lewis number for species α, which is assumed constant for each species.

2. Mixtured-averaged transport

In this model the diffusive flux is given by

$$\rho \hat{V}_{\alpha,k} Y_\alpha = -\rho D_\alpha \frac{\partial Y_\alpha}{\partial x_k} - \rho D_\alpha Y_\alpha \left(\frac{\partial \ln T}{\partial x_k} + \frac{\partial \ln \rho}{\partial x_k} - \frac{W_\alpha}{\rho R_0 T} \frac{\partial p}{\partial x_k} \right) - \rho D_\alpha Y_\alpha \theta_\alpha^{(T)} \frac{\partial \ln T}{\partial x_k}, \tag{19}$$

where the final term represents Soret effects (thermophoresis). The heat flux vector is augmented to included (final term in (20)) the Dufour effects

$$q_k = -\lambda \frac{\partial T}{\partial x_k} + \sum_\alpha \rho V_{\alpha,k} Y_\alpha h_\alpha - \sum_\alpha \frac{R_0 T}{W_\alpha} \theta_\alpha^{(T)} \left\{ \rho D_\alpha \frac{\partial Y_\alpha}{\partial x_k} + \rho D_\alpha Y_\alpha \left(\frac{\partial \ln T}{\partial x_k} + \frac{\partial \ln \rho}{\partial x_k} - \frac{W_\alpha}{\rho R_0 T} \frac{\partial p}{\partial x_k} \right) \right\}, \tag{20}$$

Here the transport properties for individual species and species pairs are evaluated using standard polynomial fits [5].

$$\ln \mu_\alpha = \sum_{l=1}^{L} c_{\mu,\alpha,l} \left(\ln \left(\frac{T}{T_{ref}} \right) \right)^{l-1} \tag{21a}$$

$$\ln \lambda_\alpha = \sum_{l=1}^{L} c_{\lambda,\alpha,l} \left(\ln \left(\frac{T}{T_{ref}} \right) \right)^{l-1} \tag{21b}$$

$$\ln D_{\alpha\beta} = \ln \left(\frac{p_{ref}}{p} \right) + \sum_{l=1}^{L} c_{D,\alpha\beta,l} \left(\ln \left(\frac{T}{T_{ref}} \right) \right)^{l-1} \tag{21c}$$

$$\theta_{\alpha\beta}^{(T)} = \sum_{l=1}^{L} c_{\theta,\alpha\beta} T^{l-1}, \tag{21d}$$

in which the $c_{\ldots,l}$ are constants, and $L = 4$. The transport properties are then obtained from combination rules following [6, 7] as

$$\mu = \left[\sum_\alpha X_\alpha \mu_\alpha^{n_\mu} \right]^{\frac{1}{n_\mu}}, \tag{22}$$

with $n_\mu = 6$,

$$\lambda = \left[\sum_\alpha X_\alpha \lambda_\alpha^{n_\lambda} \right]^{\frac{1}{n_\lambda}}, \tag{23}$$

with $n_\lambda = 1/4$, and the combination rule for molecular diffusivity follows the Hirschfelder-Curtiss approximation [8]

$$D_\alpha = \frac{\sum_{\beta \neq \alpha} X_\beta W_\beta}{W_{mix} \sum_{\beta \neq \alpha} \frac{X_\beta}{D_{\alpha\beta}}}. \tag{24}$$

Finally the combination rule for the thermal diffusion ratio is

$$\theta_{\alpha}^{(T)} = \sum_{\alpha \neq \beta} X_{\beta} \theta_{\alpha\beta}^{(T)}. \tag{25}$$

Note that the molar concentration $X_{\alpha} = Y_{\alpha} W_{mix} / W_{\alpha}$.

C. Evaluation of reaction rates

The reaction mechanism is given by M equations written in the form

$$\sum_{\alpha} \nu'_{\alpha,m} \mathcal{M}_{\alpha} \rightarrow \sum_{\alpha} \nu''_{\alpha,m} \mathcal{M}_{\alpha} \qquad \forall m \in [1, M], \tag{26}$$

where $\mathcal{M}_{\alpha}$ symbolises the chemical symbol of species α, and $\nu'_{\alpha,m}$ and $\nu''_{\alpha,m}$ are the molar stoichiometric coefficients of species α in equation m.

The reaction rate of species α is given by

$$\omega_{\alpha} = W_{\alpha} \sum_{m=1}^{M} \hat{\omega}_{\alpha,m} \tag{27}$$

where $\hat{\omega}_{\alpha,m}$ is the molar production rate of species α due to the m-th reaction, and M is the total number of reactions. The molar production rates are given by

$$\hat{\omega}_{\alpha,m} = \left(\nu''_{\alpha,m} - \nu'_{\alpha,m}\right) \left[k_{f,m}(T) \prod_{\beta=1}^{N_S} \left(\frac{\rho Y_{\beta}}{W_{\beta}}\right)^{\nu'_{\beta,m}} - k_{b,m}(T) \prod_{\beta=1}^{N_S} \left(\frac{\rho Y_{\beta}}{W_{\beta}}\right)^{\nu''_{\beta,m}} \right], \tag{28}$$

in which $k_{f,m}$ and $k_{b,m}$ are the forward and backward (respectively) rate constants for reaction m. The forward rate constant is given by the Arrhenius expression

$$k_{f,m}(T) = A_m T^{n_m} \exp\left(-\frac{E_{a,m}}{R_0 T}\right), \tag{29}$$

with A_m, n_m and $E_{a,m}$ being specified constants representing the pre-exponential factor, the temperature dependence of the pre-exponential factor, and the activation energy.

1. *Backwards rate constant*

The backwards rate constant is related to the forward rate constant via

$$\ln k_{b,m} = \ln k_{f,m} + \sum_{\alpha} \left(\nu''_{\alpha,m} - \nu'_{\alpha,m}\right) \left(\frac{\hat{g}_{\alpha}}{R_0 T} + \ln\left(\frac{P_0}{R_0}\right) - \ln T \right), \tag{30}$$

in which P_0 is a reference pressure, and $\hat{g}_{\alpha}$ is the Gibbs function for species α, which is given by

$$\frac{\hat{g}_{\alpha}}{W_{\alpha} T} = \frac{a_{\alpha,J+1}}{T} - a_{\alpha,1} \ln T + (a_{\alpha,1} - a_{\alpha,J+2}) - \sum_{j=2}^{J} \frac{a_{\alpha,j}}{j\,(j-1)} T^{j-1}, \tag{31}$$

with the coefficients again following the standard NASA polynomials [4].

2. *Third bodies*

For a step m including third bodies $\mathcal{M}$, reaction rates are scaled by the third-body concentration $c_{M,m}$, which is given by

$$c_{M,m} = \sum_{\alpha} \eta_{\alpha,m} \frac{\rho Y_{\alpha}}{W_{\alpha}}, \tag{32}$$

and $\eta_{\alpha,m}$ are the third-body efficiencies for species α for step m.

3. Lindemann forms

Where the Arrhenius form is insufficient to describe the reaction rate, a pressure (partial pressure/concentration) dependancy is introduced, and the forward reaction rate takes the Lindemann form:

$$k_{f,m} = k_{\infty,m}\left(\frac{P_r}{1+P_r}\right)F_m, \tag{33}$$

where F_m is the Troe fall-off rate, and a constant for steps with Lindemann form. P_r is a reduced pressure given by

$$P_r = \frac{k_{0,m}}{k_{\infty,m}}, \tag{34}$$

$k_{0,m}$ is the Arrhenius rate of the reaction, and $k_{\infty,m}$ is a second Arrhenius rate. Hence for Lindemann steps, in addition to the three Arrhenius coefficients, the constant F_m is prescribed, along with the pre-exponential factor, the temperature dependence of the pre-exponential factor and the activation energy for $k_{\infty,m}$. Note that after evaluation of the rate constants for Lindemann steps, the third-body concentration of the step is reset to unity. Steps with non-constant Troe fall-off rate are not yet implemented.

III. THE NUMERICAL METHODS

The spatial discretisation is based on the Local Anisotropic Basis Function Method (LABFM) in the first two dimensions, and high-order finite differences in the third dimension. LABFM has been detailed and extensively analysed in [1, 2], and the present implementation is described in [3] and we refer the reader to these works for a complete description.

To summarise, the domain is discretised in the $x_1 - x_2$ plane with an unstructured point cloud of N_{12} nodes. The discretisation in the third dimension is uniform: there are N_3 copies of each node in the $x_1 - x_2$ plane, equispaced along the x_3 axis. Hence, there are a total of $N = N_{12}N_3$ nodes in the domain. Each node $a \in [1, N]$ has position $x_{a,i}$, a distribution lengthscale s_a, and a computational stencil lengthscale h_a. s_a is the average node spacing around node a, and is analagous to the grid spacing in a finite difference scheme. The spacing in x_3 is denoted s_3, and is uniform throughout the domain, whilst s_a can vary as a function of x_1 and x_2. Each node holds the conservative variables ρ_a, $\rho u_{i,a}$, ρE_a and $\rho Y_{\alpha,a}$. The governing equations are solved on the set of N nodes.

The difference between properties at two nodes is denoted $(\cdot)_{ba} = (\cdot)_b - (\cdot)_a$. The computational stencil for each node a is denoted $\mathcal{N}_a$, and is constructed to contain all nodes b in the same $x_1 - x_2$ plane such that $|x_{i,ba}x_{i,ba}| \leq 4h_a^2$, (with the repeated subscript i implying summation), alongside all the nodes with $x_{1,ba} = x_{2,ba} = 0$ required to build a finite difference stencil in the x_3 dimension. The unstructured nodes in the $x_1 - x_2$ plane are created to fill the domain using a variation of the propagating front algorithm of [9], following [2].

A. Evaluation of spatial derivatives

Spatial derivative operators take the form

$$L_a^d(\cdot) = \sum_b (\cdot)_{ba} w_{ba}^d, \tag{35}$$

where d identifies the derivative of interest (e.g. $d = x_1$ indicates the operator L^d approximates $\partial(\cdot)/\partial x_1$, and $d = x_i x_i$ (summation implied) indicates the operator approximates the Laplacian), and w_{ba}^d are a set of inter-node weights for the operator. Note that the construction of (35) takes the same form for both LABFM-based and finite difference-based derivative approximations. For derivatives in the x_3 dimension, the weights w_{ba}^d are simply central finite difference weights of order m. For derivatives in the $x_1 - x_2$ plane, LABFM is used to set the w_{ba}^d, which is constructed from a weighted sum of anisotropic basis functions centred on node a. The basis functions used herein are formed from bi-variate Hermite polynomials multiplied by a radial basis function, and the basis function weights are set such that the operator achieves a specified polynomial consistency m. Calculation of these weights is done as a pre-processing step, and described in detail in [2].

The order of the spatial discretisation can be specified between $m = 4$ and $m = 10$, but is set by default to $m = 8$ uniformly away from boundaries. This provides a good compromise between accuracy and computational cost. At non-periodic boundaries, the consistency of the LABFM reconstruction is smoothly reduced to $m = 4$. The stencil

scale h_a is initialised to $h_a = 2.7s_a$ in the bulk of the domain, and $h_a = 2.4s_a$ near boundaries (choices informed through experience as being large enough to ensure stability). In bulk of the domain, the stencil scale is then reduced following the optimisation procedure described in [2]. This has the effect of both reducing computational costs, and increasing the resolving power of LABFM, yielding the smallest stencil size h_a for which the discretisation remains stable.

The evaluation of derivatives using (35) constitutes the bulk of the computational cost, and it is hence desirable only use it where essential, and where possible to use analytic expressions to relate derivatives of secondary properties to those evaluated via (35). First derivatives of ρ, u_i, ρE and Y_α, alongside the temperature T and pressure p, are evaluated directly using (35). Convective terms are then constructed from combinations thereof: for example

$$\frac{\partial \rho u_i E}{\partial x_i} = u_i \frac{\partial \rho E}{\partial x_i} + \rho E \frac{\partial u_i}{\partial x_i}. \tag{36}$$

Laplacians and direct second (i.e. $\partial^2 (\cdot) / \partial x_i \partial x_i$) derivatives are also evaluated with (35). To avoid the explicit evaluation of cross-second derivatives of velocity, the viscous stress divergence is evaluated as

$$\frac{\partial \tau_{ki}}{\partial x_k} = \frac{1}{\mu}\frac{\partial \mu}{\partial x_k}\tau_{ki} + \mu \left(\frac{\partial^2 u_i}{\partial x_k \partial x_k} + \frac{1}{3}\frac{\partial^2 u_k}{\partial x_i \partial x_k} \right), \tag{37}$$

where the final term is the gradient of the velocity divergence, which is calculated through two iterations of first derivative evalations with (35). The benefit of this approach is that it reduces the stencil size, and provides a significant reduction in the complexity associated with parallelising the scheme. Gradients of secondary properties are evaluated from analytic expressions in terms of gradients of primary properties. The gradient of the enthalpy of species α is given by

$$\frac{\partial h_\alpha}{\partial x_k} = \frac{dh_\alpha}{dT}\frac{\partial T}{\partial x_k} = c_{p,\alpha}\frac{\partial T}{\partial x_k}, \tag{38}$$

and the gradient of the mixture specific heat capacity is

$$\frac{\partial c_p}{\partial x_k} = \sum_\alpha \left[Y_\alpha \frac{dc_{p,\alpha}}{dT}\frac{\partial T}{\partial x_k} + c_{p,\alpha}\frac{\partial Y_\alpha}{\partial x_k} \right]. \tag{39}$$

When constant Lewis numbers are assumed, additional relations are available to evaluate the spatial derivatives of transport properties:

$$\frac{\partial \mu}{\partial x_k} = \frac{r_T \mu}{T}\frac{\partial T}{\partial x_k} \tag{40}$$

$$\frac{\partial \lambda}{\partial x_k} = \frac{r_T \lambda}{T}\frac{\partial T}{\partial x_k} + \frac{\lambda}{c_p}\frac{\partial c_p}{\partial x_k} \tag{41}$$

$$\frac{\partial \rho D_\alpha}{\partial x_k} = \frac{r_T D_\alpha}{T}\frac{\partial T}{\partial x_k}. \tag{42}$$

For the mixture averaged model, gradients of μ, λ and ρD_α are obtained directly using (35).

B. Temporal integration and filtering

The governing equations are integrated in time using a four-step third-order, low-storage explicit Runge-Kutta scheme, with embedded second order error estimation, denoted RK3(2)4[2R+]C in the classification system of Kennedy *et al.* [10]. The value of the time step is set adaptively using a PID controller to keep time-integration errors below a specified threshold, set to 10^{-4} as a default. The solution is de-aliased after each time-step by high-order spatial filtering using the filters described in [2] combined with a finite difference filter following [11]. Details of the filtering procedure can be found in [3].

C. Boundaries

For full details of the implementation of boundaries we refer the reader to [2, 3]. To summarise, the node distribution is locally structured at the boundaries, in the vicinity of which derivative operators are constructed from combinations of LABFM and asymmetric finite differences. Numerical boundary conditions are imposed through the Navier-Stokes Characteristic Boundary Condition formalism, details of which can be found in [12–14]. For walls and hard inflows, we follow the procedures described in [12]. For non-reflecting outflows we follow the approach of [13]. For non-reflecting inflows we use the *improved* formulation of [14].

D. Parallelisation

The code is accelerated with MPI and OpenMP, and uses a static non-uniform block-structured three-dimensional domain decomposition. The distributed parallelism with MPI scales extremely well ($> 96\%$ from 4 to 1024 MPI ranks). The shared parallelism is limited by memory access competition, and scales very poorly - improvements are in the pipeline here. For now, we use 1 thread, and the code is hard-coded to use only 1 thread by default.

In the x_1 and x_3 directions, periodic boundary conditions are imposed through the MPI structure, whilst in the x_2 direction they are imposed locally on each MPI rank. Communications are blocking, and due to the cyclical structure for any domain periodicity, **an even number of MPI ranks is required in each direction**.

IV. THE CODE

There are currently four branches of the code, which are becoming increasingly divergent, but they are summarised as follows:

1. `main` - the main branch which solves the equations above in dimensional form for multistep chemistry.
2. `nondim` - this branch solves for single-step chemistry in entirely non-dimensional form.
3. `oned` - this is a one-dimensional version which is suitable for quickly simulating laminar flames to generate input files for the main branch.
4. `ev` - solves the isothermal Navier-Stokes equations for a single species, along with a constitutive equation for viscoelastic flows. This branch is unrelated to combustion. Details of the equations and formulations for this branch can be found in [15].

Check github regularly for the most up to date versions. Please do not directly modify any of the above branches (especially main). Create a new branch to work on, and we will merge developments into `main` later.

A. Directory structure

The code is in a directory called `sunset_code`, and has a structure as follows:

- `data_out` - contains
  - files called `nodesxxxxx`, where `xxxxx` is equal to 10000 plus the processor number associated with that node file. (e.g. the nodes file saved by processor 4 is called `nodes10003`).
  - files called `fieldsxxxxx_xxx`, where `xxxxx` again indicates the processor, and `_xxx` gives the number of the output (e.g. `fields10003_123` is the 123^{rd} output of the 4^{th} processor.
  - `time.out`, which lists the output fields files and the number of nodes in each file.
  - a directory `statistics` which contains a set of files holding the temporal variation of various global statistics.
- `docs` - contains this document and associated latex files.
- `obj` - contains the object/mod files generated on compilation.
- `paraview_files` - holds .vtu files (once produced by the code in `vtk_conv`) which can be loaded into Paraview.

- `runfiles` - contains subdirectories each holding template control files for various levels of chemistry.
- `restart` - contains files needed to restart the simulation from a previous run.
- `source` - contains the source code for the main program in directories `base` and `common`, and source code used to generate input discretisations in `gen` and `ishift`.
- `vtk_conv` - contains a small code written in Fortran which converts the `fieldsxxxxx_xxx` into .vtu files in the `paraview_files` directory.

B. Code structure

The code has a modular stucture, with each module (each file in `source/base/`) containing routines which perform similar operations, or which perform operations in a similar context. For example, routines which evaluate spatial derivatives are located in `source/base/derivatives.F90`, and routines which evaluation thermodynamic quantities (e.g. T from ρE, ρu_i and Y_α, or c_p from T) are contained within `source/base/thermodynamics.F90`.

At the high level, the code has the following structure. The main program is in `source/base/sunset.F90`, which

- Calls routines to setup the domain, domain decomposition, LABFM and FD operators, and initial/restart conditions.
- Initialises value of time-step δt, then starting at $t = 0$, loops whilst $t < t_{max}$
  - Calls output routines if it is the correct time to output data
  - Calls routines to evaluate the time-step δt.
  - Calls a routine in `source/base/step.F90` which conducts one full RK time step, consisting of
    - Calling routines in `source/base/rhs.F90` to evaluate the RHS of the equations
    - Integrating forward one RK substep
    - Re-applying BCs
    - MPI communications
    - Repeating the above 4 steps for all RK substeps
    - Filtering the solution
    - Re-applying BCs
    - MPI communications
- Cleans up and ends the simulation.

The routines in `source/base/rhs.F90` call other routines to evaluate thermodynamic and transport properties, derivatives, chemical source terms, and characteristic boundary conditions.

C. Prerequisites

The code is largely free of dependencies on external libraries, with the exception of `openblas`, which is used for solving the local consistency matrices for LABFM. An in-house SVD routine has been tested and implemented for systems where `openblas` is not available, but the performance is better with `openblas`, and the SVD routine is not included by default.

It also requires a system with `gfortran` (for non-MPI runs, and compilation of the codes in `vtk_conv`, `source/gen/` and `source/ishift`), `mpifort` (for MPI runs), and the OpenMP libraries. These requirements should be trivial to satisfy for most Unix systems. For other compilers, modify the `Makefile` as required.

V. RUNNING THE CODE

A. Generating the discretisation

The first step in running the code is to generate a node discretisation for the desired geometry. This is done by navigating to `source/gen/`, modifying `datclass.F90`, compiling it (just use `make`), and running it (using `.gen2D`).

This little code has a number of cases (e.g. doubly periodic, periodic channel, simple flame-tube, inflow-outflow with obstacle/bluff body) which you can choose at runtime. More can be added. Changing geometry and parameters controlling the resolution etc is done by hard-coding and recompiling for each case. This code uses the propagating front algorithm (which some tweaks) of [9] to generate the distribution. See Appendix A for more (important) details.

The next step is to prepare the discretisation for the main `sunset-code`. To do this, navigate to `source/ishift/` and compile the code herein (use `make`). First, run `./ishift` and enter 1 at the prompt. This will perform 50 iterations of a simple shifting procedure, which makes the node distribution slightly smoother and better suited to LABFM.

Next re-run the `./ishift`, but at the prompt enter 0. The next prompts will request the number of MPI processors in the x, y and z (i.e. x_1, x_2 and x_3 directions). For 2D simulations choose 1 for the number in z. In general, if the number of processors in any direction is not equal to 1, it should be **even**. Otherwise there is a risk the code will get stuck on a blocking MPI communication and just hang. For example, to run a two-dimensional simulation with 24 MPI processors, we would run `./ishift`, then enter 0, 6, 4, 1.

Note that care must be taken to ensure the number of processors in any direction is not too large for the resolution in that direction. For the above example with a 6 by 4 decomposition schedule, if the domain is a very long narrow flame-tube, the sub-domains will be too narrow (smaller than the computational stencil), and the `sunset_code` will crash. The code `ishift` provides a warning if the decomposition schedule is likely to result in MPI domains which are too small, but it is not foolproof. In general, for domains with significantly varying resolution, it is harder to find a decomposition schedule which doesn't cause these problems.

With the above steps complete, a file called `IPART` will be placed in the main directory. This file contains the decomposition schedule and a list of all the nodes, and is read in by `sunset_code` at the start of the simulation.

B. Control

The simulation is then controlled by three files, all with the suffix `.in`: `control.in`, `thermochem.in` and `transport.in`. These contain the main control parameters, the thermochemistry of the fluids being simulated, and the transport properties (respectively). The latter two are less likely to require editing.

Clean copies of all three files are located within the sub-directories of `runfiles/` for the different chemistries available. Note, formatting of these files is sensitive: if you add an erroneous blank line to any of these files, `sunset_code` will read it incorrectly.

The parameters in `control.in` are largely self-explanatory, but more detail will be added to this document in due course.

1. Setting the initial flow field

This is partially controlled by `control.in`, for example in the specification of a base-flow profile (zero, uniform or parabolic), the choice of flow type (e.g. flame, Gaussian hot-spot, loading a flame file), and coordinates/scales of initial flames. The routines which implement the initial flow state are in `source/base/setup_flow.F90`, and the routines which implement error-function based flame profiles, Gaussian hotspots, density stratification etc can be modified as required.

C. Running from initial conditions

- Run the above steps to generate `IPART`
- Copy the control files from the appropriate sub-directory with `runfiles` and modify as required.
- Modify `source/base/setup_flow.F90` to specify the required initial conditions.
- Compile sunset code using (e.g.) `make react=1`
- Run the code on the correct number of processors using (e.g.) `mpirun -n 24 ./sunset`

D. Running from a restart file

- Copy `data_out/nodes*` to `restart/.`.
- Copy `data_out/*xxx` to `restart/.`, where xxx indicates the number of the output file from which you wish to restart.
- Within `restart`, run `sh fnm_strip.sh` $a+1$ times, where a is the number of digits in xxx above. This removes the underscore and all subsequent characters from the `fieldsxxxxx_xxx` filenames, ready for the code to load them on restart. e.g. if you're restarting from output file $xxx = 123$, run `sh fnm_strip.sh` 4 times.
- If the previous run was a restart, you're ready to go. Otherwise, store any output files you wish to save, then in the main directory, `make clean`, then (e.g.) `make react=1 restart=1`.
- Run the code, using (e.g.) `mpirun -n 8 ./sunset`

Note that the file `IPART` in the main directory for a restarted run needs to be the same (exactly the same) as the one used for the original run. It is not (at present) possible to change the decomposition schedule between original and restarted runs.

E. Screen output

If running the code interactively (i.e. not via a grid-engine/jobscript/queue system), it is best to use a terminal 80×24 characters. Every 100 time-steps the screen will be updated with information on the simulation, including maxima and minima for density, velocity, pressure and temperature, the time-step, and some information on profiling and performance.

Appendix A: Details on generating a discretisation

Within each case in `datclass.F90` there are several steps. Figure 1 shows a snippet of code for an example case, of a porous geometry (an array of cylinders) with inflow and outflow conditions. Running through the case in order, we have:

- Lines 195 to 200: specifying the domain size `xl` and `yl`, and other lengthscales relevant to the geometry. In this case `D_cyl` is the cylinder diameter, `S_cyl` is the cylinder spacing, `h0` is the cylinder radius, and `dx0` is the *baseline* resolution.
- Line 201: specifying the non-NSCBC boundaries with `xbcond` and `ybcond`. Values of 1, 2 and 3 indicate periodic, symmetry and low-order walls respectively. A value of 0 is used where the boundary is going to be set by the NSCBC (i.e. for isothermal/adiabatic walls, or inflow/outflow conditions).
- Lines 203 to 210: Creating the outer geometry. The domain is a rectangle, defined by 4 boundary patches connection the 4 nodes specified in lines 207 to 210. The entries in the array `b_type` define the type of boundary on the respective patch. Values of 0, 1 and 2 indicate isothermal/adiabatic walls, inflows, and outflows respectively. A value of 3 indicates the non-NSCBC boundary conditions defined by `xbcond` and `ybcond` are to be used. In this example, the first and third patches (lower and upper) have periodic conditions, whilst the second (right) and fourth (left) are outflow and inflow respectively.
- Lines 211 to 231: define a set of *blobs* which are the solid obstacles making up the porous media. Each blob is defined by the coordinates of its centre `blob_centre`, a set of coefficients `blob_coeffs`, and a rotation `blob_rotation`. The shape of the blob is then defined in polar coordinates (about the its centre) as

$$r_{blob}(\theta) = \sum_{n=1}^{\texttt{n\_blob\_coefs}} a_n cos\left(2\pi n\left(\theta + \theta_0\right)\right), \tag{A1}$$

  where the a_n are the blob-coefficients, θ_0 is the blob rotation, and r_{blob} and θ are the polar coordinates with origin at the blob centre. This construction can be used to make quite a variety of shapes, provided a) the centre of area lies within the blob (i.e. no horseshoes), and b) the maximum curvature of the blob surface can be resolved by the resolution. I have a Matlab script which generates coefficients, and has been tested for rounded edge squares, triangles, and aerofoils. In the present example, the blobs are circular, so `blob_coeffs` $= \texttt{h0}, 0, 0, 0 \ldots$.

```
!! ------------------------------------------------------------------------------------------
case(7) !! Porous with in-out

     D_cyl = 1.0d0
     S_cyl = D_cyl*1.25d0
     yl = 2.0d0*S_cyl
     xl = sqrt(3.0d0)*S_cyl
     h0=D_cyl/2.0d0      !cylinder radius
     dx0=D_cyl/60.0       !75
     xbcond=0;ybcond=1

     nb_patches = 4
     allocate(b_node(nb_patches,2),b_edge(nb_patches,2))
     allocate(b_type(nb_patches))
     b_type(:) = (/ 3, 2, 3, 1/)
     b_node(1,:) = (/-2.250d0*xl, -0.5d0*yl /)
     b_node(2,:) = (/4.0d0*xl, -0.5d0*yl /)
     b_node(3,:) = (/4.0d0*xl, 0.5d0*yl /)
     b_node(4,:) = (/-2.25d0*xl, 0.5d0*yl /)
     open(unit=191,file="blob_fcoefs.in")
     read(191,*) n_blob_coefs
     nb_blobs = 7
     allocate(blob_centre(nb_blobs,2),blob_coeffs(nb_blobs,n_blob_coefs),blob_rotation(nb_blobs))
     do i=1,n_blob_coefs
        read(191,*) blob_coeffs(1,i)
        blob_coeffs(:,i) = blob_coeffs(1,i)
     end do
     blob_centre(1,:) = (/0.0d0,0.0d0/)   !! Row 0
     blob_centre(2,:) = (/0.0d0,-S_cyl/)
     blob_centre(3,:) = (/0.0d0,S_cyl/)
     blob_centre(4,:) = (/S_cyl*sqrt(3.0d0)/2.0d0,-0.5d0*S_cyl/) !! Row 1/2
     blob_centre(5,:) = (/S_cyl*sqrt(3.0d0)/2.0d0,0.5d0*S_cyl/)
     blob_centre(6,:) = (/-S_cyl*sqrt(3.0d0)/2.0d0,-0.5d0*S_cyl/) !! Row -1/2
     blob_centre(7,:) = (/-S_cyl*sqrt(3.0d0)/2.0d0,0.5d0*S_cyl/)

     do i=1,nb_blobs
        blob_coeffs(i,:) = 0.0d0;blob_coeffs(i,1) = 1.0d0
        blob_coeffs(i,:) = blob_coeffs(i,:)*h0
        blob_rotation(i)=-0.0d0/9.0d0
     end do

     dxmin = dx0/2.0d0
     dx_wall=dxmin;dx_in=3.0d0*dx0;dx_out=1.0d0*dx0  !! dx for solids and in/outs...!! Ratio for scaling far field...

```

FIG. 1. Code snippet for an example case.

- Lines 233 and 234: specify the resolution variation (within the hard-coded constraints). `dxmin` is the smallest resolution in the domain, and is automatically used at blob surfaces. `dx_wall`, `dx_in` and `dx_out` are the resolutions at walls, inflows and outflows on the outer geometry. There are some contrivances in this section - it has not been well automated - and for some cases additional scaling factors are included to (e.g.) stretch regions of fine resolution where a flame is likely to occur.

1. Troubleshooting

If after running `gen2D` you run `ishift` and get a segmentation fault (often stating an index of `ij_count` is out of range), it's likely the discretisation generated by `gen2D` needs modifying. The (current) best approach is to generate a discretisation with `gen2D`, run `ishift` twice, then run `sunset-code` and use paraview to visualise the first output file.

[1] J. King, S. Lind, and A. Nasar, Journal of Computational Physics **415**, 109549 (2020).
[2] J. King and S. Lind, Journal of Computational Physics **449**, 110760 (2022).
[3] J. King, Computer Methods in Applied Mechanics and Engineering **421**, 116762 (2024).
[4] S. Gordon and B. J. McBride, *Computer program for calculation of complex chemical equilibrium compositions, rocket performance, incident and reflected shocks, and Chapman-Jouguet detonations*, Tech. Rep. SP273 (NASA, 1971).
[5] R. J. Kee, G. Dixon-Lewis, J. Warnatz, M. E. Coltrin, and J. A. Miller, *A Fortran computer code package for the evaluation of gas-phase multicomponent transport properties*, Tech. Rep. SAND86-8246 (Sandia National Laboratories, 1986).
[6] A. Ern and V. Giovangigli, *Multicomponent transport algorithms* (Springer Science & Business Media, 1994).
[7] A. Ern and V. Giovangigli, Journal of Computational Physics **120**, 105 (1995).

[8] J. O. Hirschfelder, C. F. Curtiss, and R. B. Bird, *Molecular theory of gases and liquids* (1964).
[9] B. Fornberg and N. Flyer, Computers & Mathematics with Applications **69**, 531 (2015).
[10] C. A. Kennedy, M. H. Carpenter, and R. Lewis, Applied Numerical Mathematics **35**, 177 (2000).
[11] C. A. Kennedy and M. H. Carpenter, Applied Numerical Mathematics **14**, 397 (1994).
[12] T. Poinsot and D. Veynante, *Theoretical and numerical combustion* (RT Edwards, Inc., 2005).
[13] J. C. Sutherland and C. A. Kennedy, Journal of Computational Physics **191**, 502 (2003).
[14] C. S. Yoo and H. G. Im, Combustion Theory and Modelling **11**, 259 (2007).
[15] J. R. C. King and S. J. Lind, "A mesh-free framework for high-order simulations of viscoelastic flows in complex geometries," (2023), arXiv:2312.12996 [physics.flu-dyn].