Module Interface Specification for SynthEddy

Phil Du (Software)
Nikita Holyev (Theory)

April 15, 2024

1 Revision History

Date	Version	Notes
2024-03-18	1.0	Initial MIS
2024-03-21	1.1	Feedbacks by domain expert addressed

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at GitHub repo

Contents

3 Introduction

The following document details the Module Interface Specifications for SynthEddy, a software to artificially generate flow field that mimics turbulent flow, which can be use as starting point for CFD simulation.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at SRS, MG.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol := is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | ... | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by SynthEddy.

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
boolean	$\mathbb{B}$	true or false

The specification of SynthEddy also uses some derived data types:

- **Array** (sequence): lists filled with elements of the same data type
- **Record**: a collection of fields, where each field is a key-value pair
- **String** (**str**): a sequence of characters

4.1 Variable Name Traceability

To help program development and understanding, some notations regarding the eddy profile from the SRS TM and DD are altered in the MIS to better reflect their usage in the code:

- `sigma`: length scale of eddy, `length_scale` in eddy profile (σ) [SRS: TM1].
- `alpha`: eddy intensity vector ($\boldsymbol{\alpha}$) [SRS: DD3].
- `intensity`: magnitude of `alpha` vector, used in eddy profile.

4.2 Abstract Data Types

As several modules listed in Section 5 are Abstract Data Types (ADTs), this documents use their types as follows:

- **VectorT**: 3-element NumPy array $\mathbb{R}^3$, representing a 3D position or velocity vector and vector/matrix operation methods [MG: M9].
- **EddyProfileT**: Eddy profile object, stores a Record of different types of eddies with their parameters (intensity, length_scale), and weights for random generation [MIS8].
- **FlowFieldT**: Flow field object, stores all EddyT objects in that field, with methods for velocity sum calculation, see Flow Field Module [MIS9].
- **QueryT**: Query interface object, handles the request to query a field, see Query Interface Module [MIS7].

Additional Abstract Data Types:

- **Figure**: matplotlib figure object, for visualization.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	**Level 2**
Hardware-Hiding Module	
Behaviour-Hiding Module	Main Control Module Query Interface Eddy Profile Module Flow Field Module Eddy Module Shape Function Module
Software Decision Module	File I/O Module Vector Module Visualization Module

Table 1: Module Hierarchy

6 MIS of Main Control Module

6.1 Module

`main`

6.2 Uses

- Query Interface [MIS7]
- Flow Field Module [MIS9]
- Eddy Profile Module [MIS10]

6.3 Syntax

6.3.1 Exported Constants

None.

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
`new`	`profile_name` str, `field_name` str, `dimensions` $\mathbb{R}^3$, `avg_vel` $\mathbb{R}$	-	unrecognized arguments, required args not inputted
`query`	`field_name` str, `query_name` str, `shape_func_name` str, `cutoff` $\mathbb{R}$	-	unrecognized arguments, required args not inputted

6.4 Semantics

6.4.1 State Variables

- `profile`: EddyProfileT, the eddy profile object to be used for generating the flow field.
- `field`: FlowFieldT, the flow field object to be generated.
- `query`: QueryT, the query interface object to handle the request.

6.4.2 Environment Variables

- `console`: str, the command line console display.

6.4.3 Assumptions

None.

6.4.4 Access Routine Semantics

`new(profile_name, field_name, dimensions, avg_vel)`:

- transition:
  - `profile` := `eddy_profile.load(profile_name)` load the eddy profile.
  - `field` := `flow_field.init(profile, field_name, dimensions, avg_vel)` create a new field.
  - Call `flow_field.save()` to save itself.
  - `console` := "Field (`field_name`) generated and saved."
- exception: exc := ($\nexists$ `profile_name` $\vee$ $\nexists$ `field_name`$\vee$ $\nexists$ `dimensions`$\Rightarrow$ "Required arguments not inputted"). `avg_vel` has default value of 0.0 so it can be omitted.
- exception: exc := (input $\notin$ {`profile_name`, `field_name`, `dimensions`, `avg_vel`} $\Rightarrow$ "Unrecognized arguments")

`query(field_name, query_name, shape_func_name, cutoff)`:

- transition:
  - `field` := `flow_field.load(field_name)` load a saved field.
  - `query` := `query.init(field)` initialize the query interface with the field.
  - if `shape_func_name` is passed, call `shape_func_nametion.set_active(shape_func_name)` to set the active shape function.
  - if `cutoff` is passed, call `shape_func_nametion.set_cutoff(cutoff)` to set the cutoff value.
  - `console` := `query.handle_request(query_name)` to get the velocity vectors at the queried positions and times specified in `query_name`, and output an operation summary (where the raw result and plot are saved).
- exception: exc := ($\nexists$ `field_name` $\vee$ $\nexists$ `query_name` $\Rightarrow$ "Required arguments not inputted").
- exception: exc := (input $\notin$ {`field_name`, `query_name`, `shape_func_name`, `cutoff`} $\Rightarrow$ "Unrecognized arguments")

6.4.5 Local Functions

None.

7 MIS of Query Interface Module

7.1 Module

`query`

7.2 Uses

- Flow Field Module [MIS9]
- Visualization Module [MIS13]
- File I/O Module [MIS12]

7.3 Syntax

7.3.1 Exported Constants

None.

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
`init`	`field` FlowFieldT	-	-
`handle_request`	`query_name` str	`response` str	`InvalidRequest`

7.4 Semantics

7.4.1 State Variables

- `field`: FlowFieldT, the flow field object to be queried.
- `request`: Record from request File
  - `mode`: str, query mode, ("meshgrid" or "points").
  - `params`
    - (meshgrid mode) `low_bounds`: VectorT, lower bound of the meshgrid.
    - (meshgrid mode) `high_bounds`: VectorT, upper bound of the meshgrid.
    - (meshgrid mode) `step_size`: $\mathbb{R}$, step size of the meshgrid.
    - (meshgrid mode) `chunk_size`: $\mathbb{N}$, grid size in each chunk.
    - (points mode) `coords`: $\{$VectorT$\}$, array of points to query.
    - `time`: $\mathbb{R}$, time to query.
  - `plot` (only in meshgrid mode to save a slice cross-section plot)

    * `axis`: str, axis perpendicular to plot slice ("x", "y", "z").
    * `index`: $\mathbb{N}$, index along the axis to get the slice.
    * `size`: $\mathbb{N}^2$, pixel size of the saved image.

- `velocities`: array of VectorT velocities at the queried positions.
- `figure`: figure object outputted by Visualization Module (if requested).

7.4.2 Environment Variables

- `query_file`: JSON file containing the query, named after `query_name`.
- `result_file`: NumPy (.npy) file containing the raw computing result of the query.
- `plot_file`: PNG file containing the visualizing plot, if requested.

The reason behind using a JSON file for the query is related to the expected use case. For large meshgrid, each run will likely be submitted to a cluster. The JSON file allow users to easily pre-define the query, instead of having to input it in command line each time. The JSON file can also be easily generated by other scripts or programs. Directly passing a JSON string is also supported if this module imported by other programs.
At current stage, the program does not do much after obtaining the raw result, other than saving it to a file. In the future, more processing and analysis can be added.

7.4.3 Assumptions

None.

7.4.4 Access Routine Semantics

`init(field)`:

- transition: `field` := as inputted

`handle_request(query_name)`:
Currently, points mode is a placeholder. An array of points is handled as many single point meshgrids.

- transition:
    - `request` := `file_io.read(query_name)` parse `query_file` into a Record.
    - `velocities` := `field.sum_vel_mesh(request.params)` get the velocity vectors at the queried meshgrid and time.
    - `result_file` := `file_io.write('results', velocities)` save the result.
    - `figure` := `visualize.plot_mesh(velocities, plot)` render plot if requested.

    - `plot_file` := `file_io.write('plots', figure)` save the plot if requested.

- output: `response` := str, operation summary, where the raw result and plot are saved.
- exception: exc := (`request` is not a Record or does not have expected parameters ⇒ `InvalidRequest`)

More request methods to be implemented in the future.

7.4.5 Local Functions

None.

8 MIS of Eddy Profile Module

8.1 Module

`eddy_profile`

8.2 Uses

- File I/O Module [MIS12]

8.3 Syntax

8.3.1 Exported Constants

None.

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
`init`	`profile_name` str	-	`InvalidProfile`
`get_density`	-	`densities` array of $\mathbb{R}+$	-
`get_length_scale`	-	`length_scales` array of $\mathbb{R}+$	-
`get_intensity`	-	`intensities` array of $\mathbb{R}+$	-

8.4 Semantics

8.4.1 State Variables

- `name`: str, name of the eddy profile (is also filename).
- `variants`: array of Records, each containing {`density`, `length_scale`, `intensity`}
  - `density`: $\mathbb{R}+$, how many eddies in a unit volume.
  - `length_scale`: $\mathbb{R}+$, the length scale (σ) of the eddy variant.
  - `intensity`: $\mathbb{R}+$, the intensity magnitude ($|\alpha|$) of the eddy variant.

8.4.2 Environment Variables

- `profile_file`: JSON file containing the eddy profile, named after `profile_name`.

8.4.3 Assumptions

None.

8.4.4 Access Routine Semantics

`init(profile_name)`:

- transition:
  - `name` := `profile_name`
  - `variants` := `file_io.read('profiles', profile_name)` load the eddy profile from `profile_file`.
- exception: $\mathrm{exc} := ((\nexists \texttt{density} \vee \nexists \texttt{length\_scale} \vee \nexists \texttt{intensity}) \forall \texttt{variants} \Rightarrow \texttt{InvalidProfile})$
- exception: $\mathrm{exc} := (\neg \forall \{\texttt{density}, \texttt{length\_scale}, \texttt{intensity} \in \texttt{variants}\} > 0 \Rightarrow \texttt{InvalidProfile})$

`get_density()`:

- output: $\texttt{densities} := [\texttt{density} \in \texttt{variants}]$

`get_length_scale()`:

- output: $\texttt{length\_scales} := [\texttt{length\_scale} \in \texttt{variants}]$

`get_intensity()`:

- output: $\texttt{intensities} := [\texttt{intensity} \in \texttt{variants}]$

8.4.5 Local Functions

None.

9 MIS of Flow Field Module

9.1 Module

`flow_field`

9.2 Uses

- Eddy Module [MIS10]
- Vector Module [NumPy]
- File I/O Module [MIS12]

9.3 Syntax

9.3.1 Exported Constants

None.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
`init`	`profile` EddyProfileT, `field_name` str, `dimensions` VectorT, `avg_vel` $\mathbb{R}$	-	`InvalidDimensions` `InvalidAvgVelocity` `EddyScaleTooLarge`
`load`	`field_name` str	`field` FlowFieldT	-
`save`	-	-	-
`sum_vel_mesh`	`high_bounds` VectorT, `low_bounds` VectorT, `step_size` $\mathbb{R}+$, `chunk_size` $\mathbb{N}$, `time` $\mathbb{R}$	`velocities` array of VectorT	`OutOfBoundary` `InvalidStepSize` `InvalidChunkSize` `InvalidTime`

9.4 Semantics

9.4.1 State Variables

- `profile`: EddyProfileT, eddy profile to be used to generate the flow field.
- `name`: str, name of the flow field (is also filename).
- `dimensions`: VectorT, size of the flow field, with x being the axial direction, y horizontal and z vertical.

- `avg_vel`: $\mathbb{R}$, average flow velocity along x-axis.
- `N`: $\mathbb{N}$, total number of eddies in the field.
- `init_x`: array of $\mathbb{R}$, initial x-coordinates of all eddies in the field.
- `y`: set {array of $\mathbb{R}$}, y-coordinates of all eddies in the field at each flow iteration.
- `z`: set {array of $\mathbb{R}$}, z-coordinates of all eddies in the field at each flow iteration.
- `sigma`: array of $\mathbb{R}+$, length scales (σ) of all eddies in the field.
- `alpha`: array of VectorT, intensity vector ($\boldsymbol{\alpha}$) of all eddies in the field.

9.4.2 Environment Variables

- `field_file`: binary file containing the flow field object, named after `field_name`.

9.4.3 Assumptions

- External flow [SRS: A4, MG: AC2]

9.4.4 Access Routine Semantics

`init(profile, field_name, dimensions, avg_vel, eddy_count)`:
Initialize the flow field with given eddy profile. Randomly generate eddies based on their parameters and associated weights, and give them initial positions within the flow field.

- transition:
  - `profile, name, dimensions, avg_vel` := as inputted
  - `N` := $\sum$(`profile.get_density()` $\times \prod$ `dimensions`)
  - `init_x` := [`-dimensions(0)/2` $\leq$ random $\mathbb{R} \leq$ `dimensions(0)/2`] of size `N`
  - `y` := [`-dimensions(1)/2` $\leq$ random $\mathbb{R} \leq$ `dimensions(1)/2`] of size `N`, for first 3 flow iterations
  - `z` := [`-dimensions(2)/2` $\leq$ random $\mathbb{R} \leq$ `dimensions(2)/2`] of size `N`, for first 3 flow iterations
  - `sigma` := array of size `N`, the sigma (length scale) of each eddy variant is repeated in this array by its density $\times$ field volume.
  - `alpha` := array of size `N`, the magnitude of alpha (intensity) of each eddy variant is repeated in this array by its density $\times$ field volume. Then this array is multiplied by an array of random unit vector to get the alpha vectors of all eddies.
- exception:

  - exc := (any $d \in$ `dimensions` $\leq 0 \Rightarrow$ `InvalidDimensions`)
  - exc := (`avg_vel` $< 0 \Rightarrow$ `InvalidAvgVelocity`)
  - exc := (any $2 \times \sigma \in$ `sigma` $\geq$ any $d \in$ `dimensions` $\Rightarrow$ `EddyScaleTooLarge`)

`load(field_name)`:

- output: `field` := `file_io.read('fields', field_name)` load the flow field from `field_file`.

`save()`:

- transition: `field_file` := `file_io.write('fields', field)` save the field object.

`sum_vel(low_bounds, high_bounds, step_size, chunk_size, time)`:
This function calculates the velocity at each position within a queried meshgrid, by summing the influence by all nearby eddies at a given time. Due to practical considerations compared to the theoretical models in SRS, it is very hard to explain this part with only formal notations. I had to use nature language.

- First, use the queried `time` to get its corresponding flow iteration and offset. A flow iteration (`fi`) is defined as when an entire x-length of the field dimension has passed due to the average flow velocity in x-direction. The offset is the x-difference compared to the start of current iteration.

- Position array of eddy centers := `get_eddy_centers(fi)`. In this function, if the y and z positions of the input flow iteration is saved in state variables `y` and `z`, they will be returned, otherwise they will be randomly generated and stored. Then, the offset is applied to the `init_x`. Now the center positions of all eddies at the queried time are obtained.

- To satisfy conservation of mass, eddies that are partially outside the field need to be wrapped around to the other side. In the x-direction, the previous and next flow iterations are added. In the y and z directions, the current field is copied to outside of each side and diagonally (function `get_wrap_arounds()`). To save computational resources, only the eddies that are within the field or outside but touching the field are kept (function `within_margin()`).

- The queried region within the field is bounded by `low_bounds` and `high_bounds`. Using a resolution of `step_size`, this region is turned into a meshgrid. Velocity need to be calculated at each point in this meshgrid.

- To avoid repeatedly calculating influence by eddies that are far away from any given point, the field is divided into chunks of size `chunk_size` in each direction. For each chunk, only eddies that are either inside the chunk or outside but touching the chunk are considered (function `within_margin()`).

- Call `eddy.sum_vel_chunk()` for each chunk to get the velocity at each point in the chunk. This chunk velocity array is then merged to the entire meshgrid velocity array.
- output: `velocities` := array of VectorT, the velocity vectors at each point in the meshgrid.

9.4.5 Local Functions

`get_eddy_centers(flow_iteration)`:

- output: `centers` := array of VectorT, eddy center positions at the queried flow iteration. See description above.

`get_wrap_arounds()`:

- output: `centers`, `sigma`, `alpha` := the center position of each eddy and after wrapping, with its corresponding sigma and alpha. See description above.

`within_margin(value, margin, low_bound, high_bound)`:

- output: out := $\mathbb{B}$, $(\texttt{value} < \texttt{high\_bound} + \texttt{margin}) \land (\texttt{value} > \texttt{low\_bound} - \texttt{margin})$

10 MIS of Eddy Module

10.1 Module

`eddy`

10.2 Uses

- Vector Module [NumPy]
- Shape Function Module [MIS11]

10.3 Syntax

10.3.1 Exported Constants

None.

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
`sum_vel_chunk`	`centers` array of VectorT, `sigma` array of $\mathbb{R}+$, `alpha` array of VectorT, `x_coords` array of $\mathbb{R}$, `y_coords` array of $\mathbb{R}$, `z_coords` array of $\mathbb{R}$	`velocities` array of VectorT	-

10.4 Semantics

10.4.1 State Variables

None.

10.4.2 Environment Variables

None.

10.4.3 Assumptions

- All eddies are spherical (currently the only function available). Additional functions can be added to this library to support oblong/fat eddies.

10.4.4 Access Routine Semantics

`sum_vel_chunk(centers, sigma, alpha, x_coords, y_coords, z_coords)`:

- output: `velocities` := array of VectorT, the velocity vectors at each point in the chunk due to spherical influence by each eddy. See [SRS: GD1].

10.4.5 Local Functions

None.

11 MIS of Shape Function Module

11.1 Module

`shape_func_nametion`

11.2 Uses

- Vector Module [NumPy]

11.3 Syntax

11.3.1 Exported Constants

None.

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
`set_active`	`shape_func_name` str	-	FunctionNotDefined
`set_cutoff`	`cutoff_value` $\mathbb{R}$	-	InvalidCutoff
`get_cutoff`	-	`cutoff_value` $\mathbb{R}$	-
`active`	`dk` $\mathbb{R}$, `sigma` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`quadratic`	`dk` $\mathbb{R}$, `lsigma` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`gaussian`	`dk` $\mathbb{R}$, `lsigma` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-
`...`	`dk` $\mathbb{R}$, `sigma` $\mathbb{R}$	`shape_val` $\mathbb{R}$	-

User can modify this module to add more shape functions.

11.4 Semantics

11.4.1 State Variables

- `active`: the function that is currently designated as the active shape function.
- `cutoff`: $\mathbb{R}$, the cutoff value for the shape function.

11.4.2 Environment Variables

None.

11.4.3 Assumptions

- `sigma` is always passed as input. Some shape functions may not use it, but it is always assumed to be passed by the caller so that any shape function can be used interchangeably without needing to modify the caller's module.

11.4.4 Access Routine Semantics

`set_active(shape_func_name)`:

- transition: `active` := the function in this module with name `shape_func_name`, so that other modules can always call `shape_func_nametion.active()` to use the designated function.
- exception: exc := (`shape_func_name` $\notin$ {functions in library} $\Rightarrow$ `FunctionNotDefined`)

`set_cutoff(cutoff_value)`:

- transition: `cutoff` := `cutoff_value`
- exception: exc := (`cutoff_value` $\leq 0 \Rightarrow$ `InvalidCutoff`)

`get_cutoff()`:

- output: `cutoff_value` := `cutoff`

`active(dk, sigma)`:

- output: `shape_val` := $\mathbb{R}$, depending on the currently active function, because the variable `active` points to a function in this module, calling it will actually call the designated function behind it.

`quadratic(dk, sigma)`:

- output: `shape_val` := $\texttt{sigma} \times (1 - \texttt{dk})^2$ if $\texttt{dk} < 1$ otherwise 0. See [SRS: TM1]. This function is not affected by change of `cutoff` value, otherwise the output will be negative when `dk` is greater than 1.

`gaussian(dk, sigma)`:

- output: `shape_val` := $Ce^{0.5\pi\texttt{dk}}$ if $\texttt{dk} < 1$ otherwise 0, where $C = 3.6276$ (provided by Nikita)

11.4.5 Local Functions

None.

12 MIS of File I/O Module

12.1 Module

`file_io`

12.2 Uses

- Hardware Hiding Module [OS]

12.3 Syntax

12.3.1 Exported Constants

None.

12.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
`read`	`sub_dir` str, `name` str	Record or Array or FlowFieldT	FailToRead
`write`	`sub_dir` str, `name` str, `content` Record or Array or FlowFieldT or Figure	-	FailToWrite

This module takes care of file directory (always local to the program), handling specific formats and exceptions, so that other modules can focus on their own tasks.

12.4 Semantics

12.4.1 State Variables

None.

12.4.2 Environment Variables

- `dir`: str, directory of the program
- `file`: file on disk to be read or written

12.4.3 Assumptions

- Array passed is large (such as calculation result), so it is faster to save it in binary format than serializing it to format like JSON.

12.4.4 Access Routine Semantics

`read(sub_dir, name)`:

- output: out := Record, Array or FlowFieldT, depending on the type of file read.
- exception: exc := (file cannot be found or not the expected data type ⇒ FailToRead)

`write(sub_dir, name, content)`:

- transition: write the `content` to the file on disk, depending on data type.
  - Record ⇒ JSON
  - Array ⇒ NumPy binary
  - FlowFieldT ⇒ pickle binary
  - Figure ⇒ PNG
- exception: exc := (file cannot be written to disk ⇒ FailToWrite)

12.4.5 Local Functions

None.

13 MIS of Visualization Module

THIS IS A PLACEHOLDER [MG: AC5]

13.1 Module

`visualize`

13.2 Uses

- None

13.3 Syntax

13.3.1 Exported Constants

None.

13.3.2 Exported Access Programs

Name	**In**	**Out**	**Exceptions**
...	...	...	...

13.4 Semantics

13.4.1 State Variables

?

13.4.2 Environment Variables

?

13.4.3 Assumptions

?

13.4.4 Access Routine Semantics

?

13.4.5 Local Functions

?

References

Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL http://citeseer.ist.psu.edu/428727.html.

14 Appendix

[Extra information if required —SS]

15 Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Problem Analysis and Design. Please answer the following questions:

1. What are the limitations of your solution? Put another way, given unlimited resources, what could you do to make the project better? (LO_ProbSolutions)

2. Give a brief overview of other design solutions you considered. What are the benefits and tradeoffs of those other designs compared with the chosen design? From all the potential options, why did you select the documented design? (LO_Explores)