



Google Summer of Code



**GSOC'24 Project Proposal
for**

**Extending Look-Up Table
Features for Thermodynamics
and Chemistry
(Duration: 175 Hours)**

VAISHNAVI GUPTA



[VAISHNAVIGUPTA04](#)



vaishnavi.gupta.mec21@iitbhu.ac.in



[vaishnavi-gupta-08b17b229](#)



(+91) 9305272711

1. About the Contributor

1.1 Basic Information


Name: Vaishnavi Gupta

Major: Mechanical Engineering

Degree: Bachelor of Technology

Year: 3rd

Institute: Indian Institute of Technology BHU, Varanasi, India

Resume:  core_resume.pdf

Time zone: Indian Standard Time (UTC + 5.30)

1.2 About Me

I have been interested in physics since school, mainly **thermodynamics** and **Energy**. Following my interest and passion for studying these in-depth, I took **Mechanical Engineering** as my Bachelor's degree. In my course curriculum, I got the opportunity to dive further into subjects like **Thermodynamics, Fluid Mechanics, Heat and Mass Transfer**. I describe myself as a hard-working and passionate student. Currently, I am in the top 20 students of my batch. My research interest lies in **Conductive heat transfer and Computational Fluid Dynamics**. In my college life up until now, I have also tried to explore as many technical fields as possible. Fields like **Machine Learning, Artificial intelligence and robotics** have attracted me powerfully. Recently, I have developed a keen interest in implementing different machine learning algorithms to the data obtained from other CFD simulations or thermal experiments to predict helpful information.

1.3 Work Experience

I have worked as a research intern under **Prof. Anurag Goyal at the TSRG Lab Department of Mechanical Engineering, IIT Delhi**, on the project **“Thermal energy storage technologies for commercial buildings in the USA for enhanced energy efficiency and resiliency.”**

In this project, I extracted and analyzed thermal energy consumption data from the **NREL site** for heating and cooling purposes in commercial and residential buildings. I used **Python** and **Excel** for the data analysis. After cleaning and some moderation, the data was used to predict whether TES integration with heat pumps would be efficient and cost-effective. This was simulated on **EES** and **MATLAB**. During my internship, I became familiar with most of the technologies prevalent in the USA for electricity supply. I am attaching my [final report](#) to look over.

1.4 Major Projects

- **A Novel Patient Treatment Planning for Localized Cancer of Internal Organs**

In this project, I used CT scans or MRIs to create 3D models of kidney tumors and simulated Radiofrequency or microwave ablation using COMSOL Software. Applying Pennes's bioheat transfer equation, I monitored temperature distribution and tissue damage, potentially revolutionizing cancer treatment.

- **Simulation of Vortex Shedding past a Triangular Bluff Body with Secondary Airflow**

The V-gutter flame represented secondary flow, while the duct flow represented primary flow in a 2D domain. Utilizing a transient pressure-based numerical solver in OpenFOAM Software with the SST k-omega model for turbulence modeling, I observed that higher air velocity suppresses vortex-shedding, while increased temperature enhances vortex formation.

2. Project Introduction

2.1 Abstract

SU2 is a collection of open-source software tools developed in C++ for numerically solving partial differential equations (PDEs) and doing PDE-constrained optimization. We can solve a wide variety of problems in SU2, including industrial studies. For instance, fluids with higher thermal conductivity and heat retention capacity are commonly employed in waste heat recovery systems, propeller jets, and other similar devices. Using the standard ideal gas equations to solve these scenarios is quite challenging, resulting in the identification of its thermodynamic state being an extremely laborious task. The computational cost associated with fluid thermodynamic models expressed in terms of equations of state (EoS) can become a limiting factor if accurate estimations are needed in combination with expensive simulations.

A data-driven approach may be opted to ascertain the fluid's thermo-chemical condition in such a situation. To decrease the computational time related to calculating thermo-physical fluid properties while maintaining a satisfactory level of accuracy, look-up table (LuT) methods are convenient and widely adopted. The attributes of a substance in its thermodynamic states are tabulated against each other and include various parameters like temperature, pressure, specific volume, internal energy, enthalpy, and entropy. The qualities at intermediate locations between tabulated values are estimated using interpolation techniques that best suit the case.

This summarizes the project context. Currently, SU2 has implications for structured and unstructured LUT methods. I will link Python interpolation algorithms to SU2 code using py wrapper files in this project.

2.2 Motivation

For interpolating the fluid's thermodynamic state during non-ideal CFD (NICFD) simulations and combustion simulations, SU2 currently employs a method akin to a trapezoidal map. This approach was created primarily to solve two-dimensional look-up issues. While reasonably quick in interpolation, the existing process has several drawbacks, including poor memory scalability, setup complexity, and limited application scope. This restricts SU2's capacity to model complicated fluid flow in large-scale issues.

The primary goal of this project is to create an adaptable library of LUT techniques using Python libraries like Scipy or Xarray, which SU2 may utilize for CFD simulations and linkage of the custom interpolation function to SU2 codebase using a Python wrapper file. It facilitates the more straightforward setup of simulations for users by providing a variety of look-up methods.

2.3 Contributions

2.3.1 Practice Assignments

- Set up a case from scratch.
 - I generated a 2-D mesh using the gmsh software for an axisymmetric turbulent jet.
 - Wrote configuration file for the same and simulated the results using ParaView.
 - Link: [Test Results](#)
- I have also studied and analyzed results by running various tutorials for learning purposes.
 - Link: [Results](#)

3. Deliverables

1. Pre-processing of input data

- a. **Expected due date:** 5th June
- b. **Objective:** Enable regularization of the scattered reference data onto rectilinear grids for structured interpolation.
- c. **Deliverable:**
 - i. I plan to regularize the scattered reference data onto the regular grid to simplify our interpolation process. I have decided to go with
 - o **Natural Neighbor Interpolation:** Natural Neighbor Interpolation is a spatial interpolation method that estimates values at arbitrary locations based on the values available at neighboring data points. It is well-suited for irregularly spaced data, such as scattered input data of a thermodynamic fluid.

Other methods can also be tried based on various factors, such as the characteristics of the data, the underlying physical processes, and the specific requirements of the problem.

2. Searching and Interpolation

- a. **Expected due date:** 16th June
- b. **Objective: Application of an efficient Searching and Interpolation Algorithm**
- c. **Deliverable:**
 - i. I plan to use algorithms based on adaptive Cartesian mesh to increase the accuracy and decrease the number of discretization points for the thermodynamic region of interest.
 - ii. I plan to use the `scipy.spatial` modules' `cKDTree` algorithm to efficiently locate the nearest neighbors or multiple data points within the table relevant to the query point to achieve satisfactory accuracy.

We use the `query` method of the `cKDTree` to find the nearest neighbor to the query point.

```
from scipy.spatial import cKDTree
# Sample thermodynamic data points (e.g., temperature
and pressure)
data_points = np.array([[300, 1.0], [350, 1.5], [400,
2.0], [450, 2.5]])
tree = cKDTree(data_points)
query_point = np.array([[375, 2.2]])
# Find the nearest neighbor to the query point
nearest_neighbor_idx = tree.query(query_point,
k=1)[1]
nearest_neighbor_point =
data_points[nearest_neighbor_idx]
```

- iii. I will use SciPy's interpolation function '`scipy.interpolate`' for Interpolation as it is computationally efficient and suitable for real-time applications.

Example code :

```
# interpolation.py
from scipy.interpolate import interp2d

def interpolate_point(x, y, data_points_x, data_points_y,
data_values):
    """
    Interpolates a point (x, y) using Scipy's interp2d function.
    """
    # Create a 2D interpolation function
    interp_func = interp2d(data_points_x, data_points_y,
data_values, kind='cubic')
    # Evaluate the interpolation function at point (x, y)
    z = interp_func(x, y)
    return z
```

3. Wrapper integration

- a. **Expected due date:** 8th July
- b. **Objective:** Embedding the interpolation code into the SU2 code
- c. **Deliverable:** In integrating Python with SU2 through a Python wrapper interface, passing the `'interpolate_point'` function defined in the wrapper to SU2 that performs interpolation using Python libraries and then calling this function from within the SU2 C++ environment.

- i. To achieve communication of the interpolation function from the driver level to the fluid model level within the C++ code of SU2, I will create a modular architecture where the driver object in the Python wrapper will set the interpolation function, and this information will be subsequently communicated to the fluid model during initialization.

- ii. In the fluid model, I will define an interface or base class named `CFluidModel` to represent the interpolation function and its properties. I plan to make this interface such that it would define methods or properties to set and evaluate the interpolation function. I plan to use code similar to

```
// FluidModel.h - Define an interface for the fluid model
class CFluidModel {
public:
    virtual void setInterpolationFunction(/* Parameters */) = 0;
    virtual double evaluateInterpolationFunction(double x) = 0;
};
```

- iii. I will set up the `PythonWrapper` class, which will communicate with the fluid model to set the interpolation function based on the parameters the Python driver provides. The following code serves as a reference :


```
#include "CFluidModel.h"
#include <Python.h>
class PythonWrapper {
private:
    CFluidModel* CfluidModel;
public:
    void setInterpolationFunction(/* Parameters */) {
        // Set interpolation function in fluid model
        CfluidModel->setInterpolationFunction(/* Parameters */);
    }
    void initializeFluidModel() {
        // Initialize fluid model
        fluidModel->initialize();
    }
};
```

- iv. In the SU2 C++ code, I will create a class to implement the fluid model interface and provide functionality to set and evaluate the interpolation function.

```
#include "CFluidModel.h"
class MyFluidModel: public CFluidModel {
private:
    // Interpolation function parameters
    // Add any additional members as needed
public:
    void setInterpolationFunction(/* Parameters */) override {
        // Set interpolation function and parameters
    }
    double evaluateInterpolationFunction(double x) override {
        // Evaluate interpolation function
        return /* Interpolated value */;
    }
};
```

This architecture will allow flexibility in defining and setting interpolation function(s) from the Python driver and integrating them with the fluid model in the C++ codebase.

4. Setting up the SU2 Config file

- a. **Expected due date:** 19th July
- b. **Objective:** Modify and enable the SU2 config file to use the custom interpolation function.
- c. **Deliverable:**
 - i. I plan to add the Parse Config Option to the config file as it will specify the use of custom interpolation functions. I will parse this option in the SU2 codebase because it will be helpful in enabling or disabling the use of custom interpolation functions.

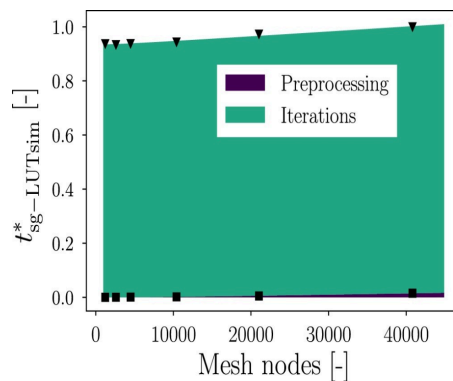
```
class SU2ConfigParser {
private:
    bool useCustomInterpolation;
public:
    void parseConfig(/* Config file */) {
        // Parse config option to enable or disable custom interpolation
function
    }
    bool isUsingCustomInterpolation() {
        return useCustomInterpolation;
    };
};
```

I will implement error-handling mechanisms in the SU2 code to handle potential errors or exceptions raised by the Python wrapper. For example, using the try block will throw an exception.

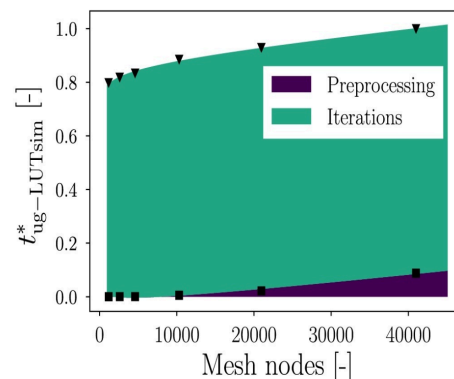
```
try {
    // Call function that may throw an exception
    Interpolation_func(/*Parameters*/);
} catch (const std::invalid_argument& e) {
    // Handle exception
    std::cerr << "Error: " << e.what() << std::endl;
}
```

5. Testing and Validation

- a. **Expected due date:** 30th July
- b. **Objective:** To ensure that everything is working fine and the results from the above interpolation method match results from the EoS method.
- c. **Deliverable:**
 - i. In order to check the proper functioning of my interpolation function, Pywrapper file and the SU2 drivers, I will be writing 2-3 Unit Testcases.
 - ii. For example, I will use my custom interpolation model to compute the flow around the NACA0012 airfoil using the ideal gas laws. Similarly, other simple cases could also be taken.
 - iii. To validate my results, I plan to solve the NICFD and combustion test cases using the traditional methods and do the same using the above interpolation algorithm.
 - iv. Then, the computation time and accuracy between the two approaches can be compared by plotting the total computation time taken by both methods. A similar comparison can be seen in the image below.



(a) *sg-LUT* CPU Time.



(b) *ug-LUT* CPU time.

6. Debugging the code

- a. **Expected due date:** 15th August
- b. **Objective:** Solve the errors that might be occurring and find appropriate solutions
- c. **Deliverable:**
 - i. I will solve and fix any errors occurring in the codebase and implement suitable changes to solve them.
 - ii. This could be done by running various test results with different approaches.

7. Documentation

- a. **Expected due date:** 26th August
- b. **Objective:** Writing Documentation for new users and opening a Pull request
- c. **Deliverable:**
 - i. I will document the test cases properly, which will be very useful for beginners and experienced users who want to understand the usage of custom interpolation functions.
 - ii. I will create and finalize a pull request enabling the users to use the custom fluid model functions using Python wrapper files.
 - iii. I plan to showcase my code and the tutorials in well-written, documented form using the doxygen software.

4. Timeline

Date	Plan
1 May- 26 May	Community Bonding Period <ul style="list-style-type: none"> • Read the documentation thoroughly • Read the details of the codebase • Join SU2 weekly meetings with mentors • Participate in group discussions • Finalize the methods and approach
27 May - 5 June	Pre-processing of input data <ul style="list-style-type: none"> • Analyze and study the input data carefully. • Learn more about the Natural Neighbour interpolation technique. • Enable regularization of input data to deal with scattered data points. • Documentation and testing
6 June - 16 June	Searching and Interpolation <ul style="list-style-type: none"> • Learn about adaptive cartesian mesh techniques. • Write code for the <code>cKDTree</code> algorithm. • Write code for the <code>scipy.interpolate</code> function.
17 June - 8 July	Py wrapper integration <ul style="list-style-type: none"> • Coding the <code>interpolate_point</code> function. • Study and get experience with the SU2 codebase. • Creating a modular architecture for the linkage. • Creating the model interface to embed the interpolation code into SU2.

	<ul style="list-style-type: none"> • Writing code for the <code>CFluidModel</code> class.
Mid-Term Evaluation	
13 July- 19 July	<p>Modifying the SU2 config file</p> <ul style="list-style-type: none"> • Study the Parse Config Option. • Writing code to change the option according to our study. • Adding the error handling code to the config file.
20 July - 30 July	<p>Testing and Validation</p> <ul style="list-style-type: none"> • Writing of Pywrapper files, Config files and other required files to set up Test cases. • Validating my results from these test cases by comparing results from other methods. • Solving the NICFD and combustion case to get another set of comparison results. • Optimizing the code to produce efficient and accurate output.
31 July- 15 August	<p>Debugging</p> <ul style="list-style-type: none"> • Analyzing the errors and outputs arising from test cases. • Modifying the code accordingly using error-handling functions
16 August - 26 August	<p>Documentation</p> <ul style="list-style-type: none"> • Using the Doxygen software, documenting the process into a clear and practical user guide. • Updating the SU2 documentation on the website by making it more beginner-friendly.
Initial GSOC'24 Results Declare	

5. Commitments

I am flexible with my working hours and can also work at night. This summer, I have no commitments as such. My summer vacation will be from 10th May to 31st July, and I can devote 5-6 hours daily. My college will resume from July onwards, but I will manage it along with my project. I will be having my mid-semester examinations in September.

Also, I check my emails at least twice a day. Communication can be done through mail and the Slack channel. I usually am a quick replier, and my response can be expected within less than 24 hours.

6. Post GSOC

As a dedicated contributor to the SU2 organization's GSoC'24 project named "Extending Look-Up Table Features for Thermodynamics and Chemistry," my commitment to the project would not end with GSoC'24. An open-source project's success depends on its community's contributions and continuous improvement. Hence, I would like to continue contributing to the project even after GSoC by staying in touch with the community, attending meetings, and collaborating with other contributors. I would ensure that the codebase remains up-to-date with the latest technologies, fix bugs, and add new features per the project's needs. I aim to keep the app functional and user-friendly while adhering to industry standards and best practices. I would work closely with the project's maintainers and other contributors to accomplish this, providing guidance and support wherever needed. Additionally, I would be available to update, debug, and maintain the code as required, ensuring it remains robust and reliable.

In summary, my commitment to the SU2 organization's GSoC'24 project named "Extending Look-Up Table Features for Thermodynamics and Chemistry" would be long-term. I would continue to contribute actively and collaboratively even after GSoC'24.