Team Number:	apmcm2307379
Problem Chosen:	B

2023 APMCM summary sheet

In this study, with the aim to optimize the temperature and wind speed distribution in a glass greenhouse to promote crop growth, we addressed four key issues step by step.

To solve the first problem, we built mathematical equations under crop-free conditions, based on the fundamental physical principles, with Python and a computational fluid dynamics software (CFD), ANSYS Fluent. The temperature and wind speed distribution at 0.5 m was further validated by the numerical simulation results from ANSYS Fluent, with an indicator (R) showing the overlapping interval, achieving 89.84% and 93.75% respectively.

For problem two, we improved the current model by adding a porous media region representing crops. The consideration on its porosity and permeability ensured us the effectiveness of R more than 89% in simulating the temperature and wind speed distribution at specific heights (0.5 m and 0.1 m) in the greenhouse, compared with results from the CFD software. An initial claim on the unsuitable conditions for crops growth was made.

For two scenarios in problem three, we adjusted the relevant parameters via the two methods, simulating the effect of individually changed fan parameter on the two target fields. Compared to results in problem two, we deducted that increasing velocity in canopy would achieve warmer living conditions in the canopy, on the contrary, lower position of the fans benefits roots for better uniformity.

In order to solve problem four, we applied genetic algorithm to optimize the number, location, speed and temperature in designing fans. With the length magnification, the constrains on number and locations of the fans and the convergence guaranteed by the Markov Chain, we could obtain relatively random schemes while making sure comfortable living conditions for crops. Two schemes including five fans and three fans were given respectively.

In summary, our research not only established an effective model to simulate the environmental conditions in the glass greenhouse, but also optimized the fan design through a genetic algorithm, providing scientific guidance and optimization solutions for greenhouse design and crop planting.

Keywords: Greenhouse simulation ANSYS Fluent Genetic algorithm

Contents

I. Introduction

Temperature, humidity and wind speed play significant importance in crop yields in modern greenhouses. To achieve the suppression of high temperature, improvement of air components, and decrement of humidity inside the greenhouse, there are two kinds of ventilation systems.

1.1 Natural ventilation

In natural ventilation systems, air flow is respectively promoted by the different temperature and different wind velocity between the inside and outside of the facility , which requires low investments but also only has limited capabilities. In summer, the greenhouse tends to rely on high-speed winds to generate horizontal air flow, while in winter, heat is more likely to be utilized with vertical air flow determined by heights between the up outlet and down inlet.

1.2 Mechanical ventilation

The mechanical ventilation systems contain two type of patterns, taking air in or exhaust.The intake ventilating system has low requirements on air tightness, could resist contamination from outside pollutants that may impair inner environment, but has a varying velocity field. On the contrary, the exhaust ventilating system provides a relatively uniform distribution of wind velocity, requires good seal, but may result in contamination from outside. The position of outlet in a mechanical system could be the at top and bottom of the greenhouse, as well as the in horizontal and longitudinal direction of the greenhouse.

II. Restatements and Analyses of problems

2.1 Restatements

Here, in order to regulate climate factors mentioned above, ventilation systems with greenhouse fans in breadth-wise are used, of which the position and the velocity speed affect the distribution and uniformity of the velocity field and the temperature field inside the greenhouse. The inlet fan blows in warm air at 40°in the horizontal direction with an average velocity of 2 m/s. The outer glass and bottom soil of the greenhouse are

set as wall conditions, primarily exchanging energy with the entire greenhouse through convective heat transfer and conduction.

Without considering external factors such as doors, drafts, solar radiation, and other environmental factors, the greenhouse that is sealed and placed indoors, should be clearly given the velocity and temperature field without (**Question one**) at a height of 0.5 meters and with crops (**Question two**) at a height of 0.5 meters and 0.1 meters respectively, analyzing the conditions for crop growth.

Further, by increasing the velocity of warm air inlet from 2 m/s to 3 m/s, and decreasing the position of the greenhouse fan by moving it from 1.3 m to 1 m, the temperature and wind speed distribution inside the glass greenhouse should be provided (**Question three**) and made comparisons with results in question two. Based on the performance of the adjustment, the number of greenhouse fan, location, wind speed, blowing temperature, specifications and different crops and other factors could then be optimized (**Question four**) for a better uniformity.

2.2 Analyses

As mentioned in the problem statements, the convective heat transfer, conduction and the aerodynamics should be considered to figure out the temperature and wind velocity field. Thus, to clearly show the two features at different heights in the greenhouse, we girded the cuboid-shaped greenhouse to limited amount of points and applied the law of conservation of energy for passing the velocity and temperature at grid point in the greenhouse with Python. To demonstrate the accuracy of our results with **Python**, we further built the 3-dimensional model in a computational fluid dynamics software, called **ANSYS Fluent**. The consistency between the mathematical model built by Python and the numerical simulation by ANSYS would then validate the reliability of the parameters optimized by our mathematical model.

III. Models

The fundamental principles and parameters that govern the velocity and temperature fields are consistent in all problems we face, which are presented in this section.

Table 1 Descriptions of Symbols

Symbols	Definitions	units
T	Temperature	°C
t	Time	s
u	Velocity vector	m/s
α	Thermal diffusivity	
p	Pressure	Pa
μ	Dynamic Viscosity	Pa ·s
Δ	Variation of certain variable	
D	Euler distance between two spatial points	m
Re	Reynolds number	
φ	Porosity	
k	Permeability	m^2

3.1 Terms, Definitions and Symbols

3.2 Assumptions

(1) The air can be considered as an incompressible fluid, and is evenly mixed in thermal equilibrium at any time.

(2) The heat transfer between greenhouse glass and soil is mainly convection and conduction, ignoring thermal radiation in our models.

(3) There is no additional heat source in the greenhouse, except for the hot, stable and uniform air blown in by the fans.

(4) The turbulence effect of air is ignored.

(5) The actual volume of crops accounts for 50% of the total volume of crops cuboid.

(6) The diameter of the grain of the porous media, and here we assume it to be the half of the height of crops.

3.2.1 *Model for transferring energy*

Based on the law of conservation of energy, a fundamental principle in physics that states that the total energy of an isolated system remains constant over time, we considered the steady-state flow with Eq. (1) [1]. Since the problems we face are non-instantaneous, without the consideration of time, in the stable stage of heat balance, we simplified the heat transferred at certain point N to Eq. (2), a function of the Euler distance between the point N and the center of the fans. Similarly, in the stable stage of velocity field, we simplified the velocity transferred at point N to Eq. (3).

$$\Delta U = Q - W \tag{1}$$

$$e^{-decay\ controller\ _T \times D_N} = Q - W \tag{2}$$

$$V_0 e^{-decay\ controller\ _V \times D_N} = V \tag{3}$$

where, $Q - W$, the difference of energy between inside and outside, is represented by the difference between initial temperature at fans and in the greenhouse. D_N denotes the distance between the center of fans and point N in the greenhouse. The decay controller is a parameter, individually adjusting the decaying effects of the temperature (T) and velocity (V) in the greenhouse. V and V_0 represents the velocity at point N and the center of the fans respectively.

By building the mathematical model at point N, we could further give the temperature field and velocity field in the whole greenhouse

3.3 Establishment of the CFD model

To test our mathematical model, we utilized ANSYS Fluent to numerically simulate the velocity and temperature field in the greenhouse. First, a 3-dimensional model of the same size as the greenhouse was established and meshed in Fig. (1).

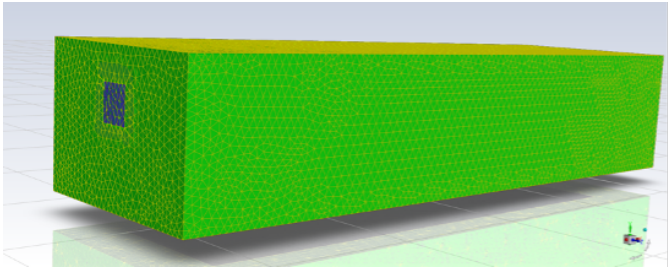


Figure 1 Grids

With the meshed grids and boundary conditions settled down, we could then simulate the target fields after the calculation of the Reynolds number in Eq. (4). The Re at the inlet is 66667, a number much larger than the smallest threshold of turbulence, 4000. So, via the finite difference method, we could simulate the realistic fields while reaching to the stable and balanced stage under the condition that air flow in the greenhouse could be regarded as turbulent motion.

$$Re = \frac{\rho V_0 L}{\mu} \tag{4}$$

where, ρ denotes the fluid density. L represents the size of the fans. μ is the dynamic viscosity of the air.

3.4 Evaluation

To evaluate the simulation accuracy between the mathematical model implemented by Python and the numerical model calculated by ANSYS, we devise a indicator, R, shown in Eq. (5).

$$R = \frac{A \cap B}{max(A, B)} \tag{5}$$

where, A and B represent the magnitude of simulation results at the same area for Python and ANSYS respectively. $A \cap B$ gives the overlapping interval, while max(A,B) is the larger interval. So, R denotes the similarity of simulation results between the two basic models, which means a high value of R suggests a good simulation accuracy, validated by both models.

IV. Solutions and Results

4.1 Question one

To realize the mathematical model we built in section 3, we first meshed grids with limited spatial points ($40 \times 20 \times 20$), corresponding to sequence of $10m \times 3m \times 2m$. The decay controller was applied to the calculation of temperature, velocity and the length of the greenhouse. Via Appendix. 1, the temperature and velocity fields in the greenhouse were shown in Fig. (2), and the distribution of wind speed and temperature at a cross-section of the greenhouse at a height of 0.5 meters were displayed in Fig. (3).

Meanwhile, the longitudinal results given by ANSYS Fluent, showing the variation of temperature and velocity in the direction of the length of the greenhouse were first

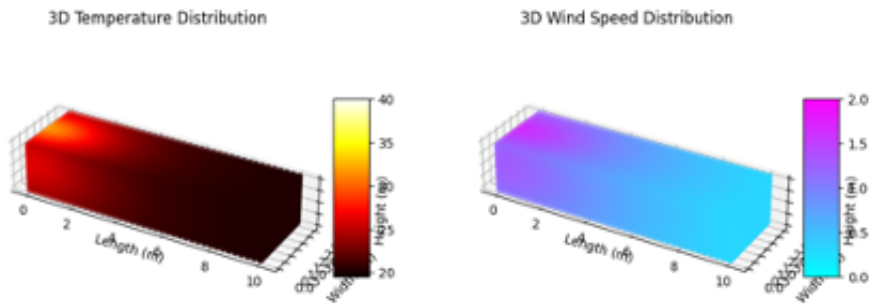


Figure 2 Three-dimensional results of the whole greenhouse in Python

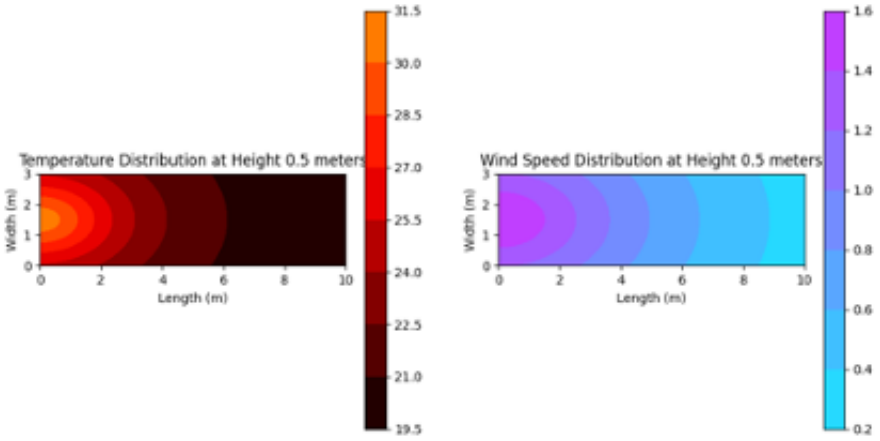


Figure 3 Results of target fields at a height of 0.5 meters in Python

exhibited in Fig. (4) and (5) respectively. Further, the distribution of wind speed and temperature at a cross-section of the greenhouse at a height of 0.5 meters were displayed in Fig. (6) and Fig. (7).

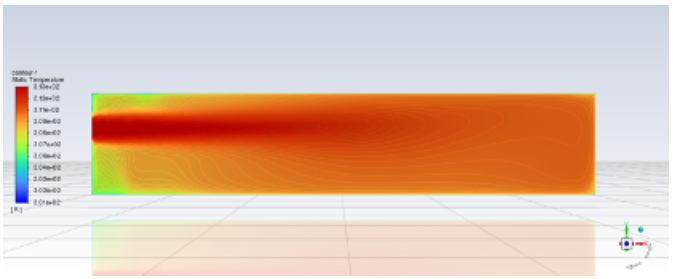


Figure 4 Longitudinal temperature simulated by ANSYS

The indicator we designed before were calculated here. The magnitude of temperature in the height of 0.5m ranged from 31.85°C to 37.85°C in our mathematical model, while that of ANSYS Fluent varied from 31.2°C to 37.6°C. So, R (6)was exemplified

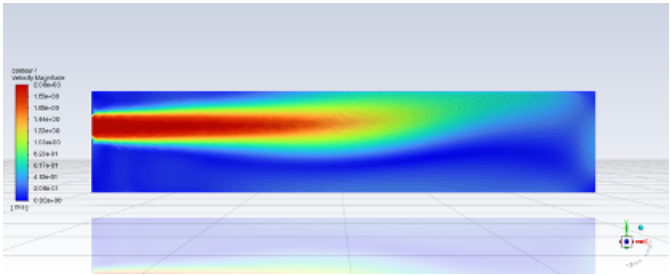


Figure 5 Longitudinal velocity simulated by ANSYS

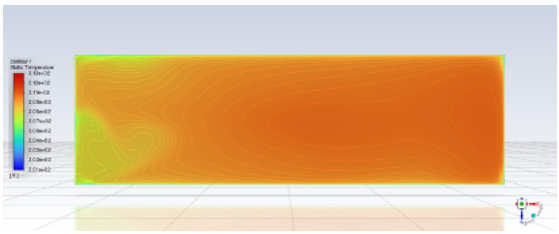


Figure 6 Distribution of temperature at a height of 0.5 meters simulated by ANSYS

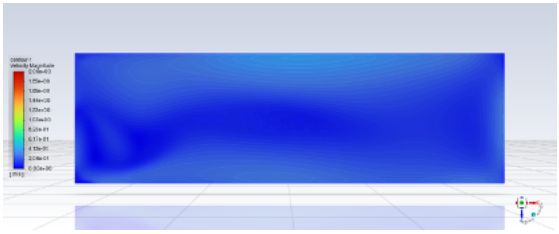


Figure 7 Distribution of velocity at a height of 0.5 meters simulated by ANSYS

in the temperature simulation.

$$R = \frac{37.6 - 31.2}{37.85 - 31.85} = 89.84\% \tag{6}$$

Similarly, we got R value of velocity, reaching 93.75%, with a range of 0-0.48m/s and 0-0.45m/s for the mathematical model and ANSYS respectively, which showed that the temperature and wind speed distributions of the two models have overlapping intervals at multiple locations. This finding demonstrates that although the mathematical model has limited ability to simulate physical details compared to CFD software, it still has some accuracy and reliability in predict the temperature and wind speed distribution in glass greenhouses. As a result, it could be a tool for quick preliminary analysis, while the numerical model via CFD software is suitable for more detailed and precise analysis. By combining these two models, it is possible to understand and optimize the climate regulation of glass greenhouses more comprehensively.

4.2 Question two

4.2.1 *Results*

Models for question two were similar to what we have shown in section 3 and section 4.1, however, differences exist and actually lie in the addition of crops[2]. So, we regarded the crop as a kind of porous media, and scaled the temperature in Eq. (2) by 0.9 and velocity in Eq. (3) by 0.5, when points were detected in the area of crops, to show the resistance of crops to the hot wind. The whole simulation results (see Appendix. (2)) were shown in Fig. (8), and the distribution of wind speed and temperature at two cross-sections within the greenhouse were shown in Fig. (9).

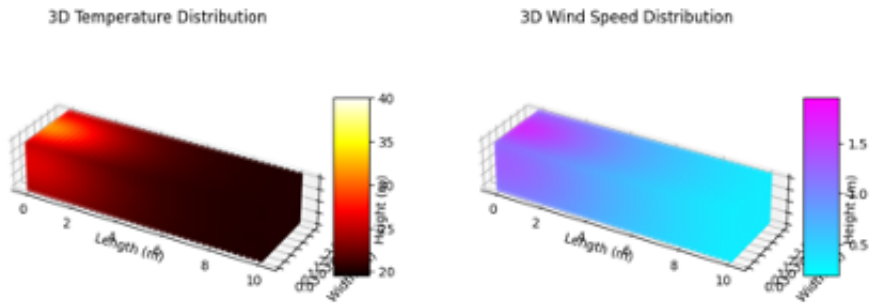


Figure 8 Three-dimensional results of the whole greenhouse with crops in Python

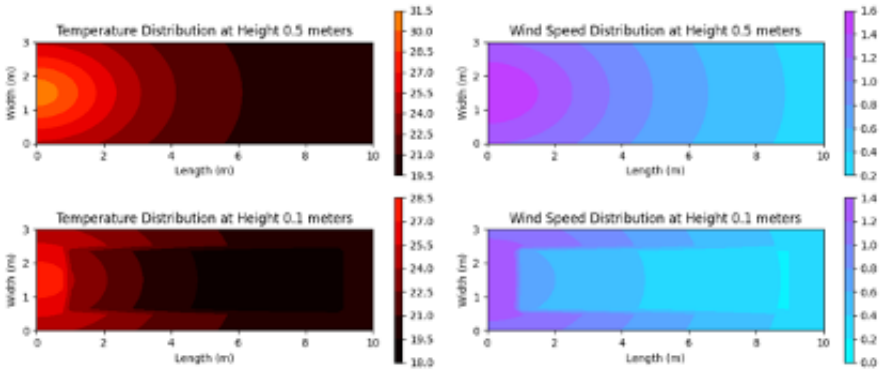


Figure 9 Results of target fields at the height of 0.5 meters and 0.1 meters in Python

Likewise, we also established the glass greenhouse with planted crops in ANSYS Fluent in Fig. (10). However, the porosity defined in Eq. (7) and permeability defined in Eq. (8) of the porous media required prior settings.

$$\varphi = \frac{V_{crops}}{V_t otal} \tag{7}$$

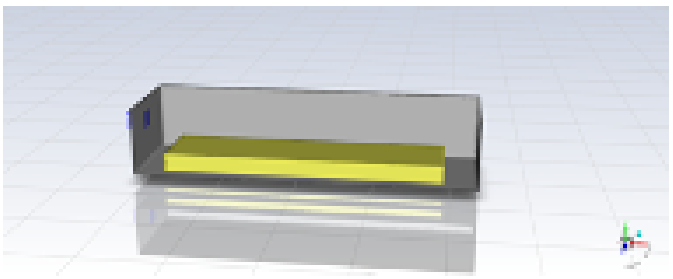


Figure 10 Three-dimensional model for the greenhouse with crops in ANSYS

where, the V_{crops} is hard to accurately assess without realistic conditions, so here in accord with the fifth assumption in section 3.2, we assumed the V_{crops} accounts for 50% of V_{total}, which ensures the value of φ to be 0.5.

$$k = \frac{d_p^2 \varphi^3}{K(1-\varphi)^2} \tag{8}$$

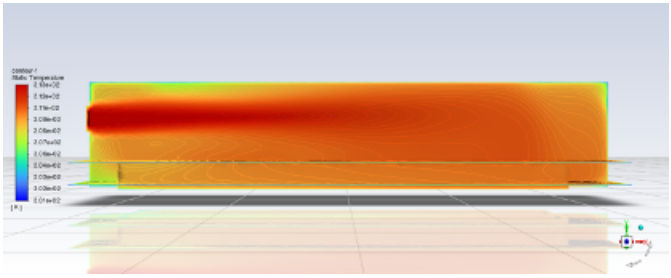
Permeability is a physical quantity that measures the flow ability of a fluid through a porous medium, typically depending on the size, shape, and distribution of pores. For a simplified crop model, we can use the Kozeny-Carman equation to estimate its permeability, where d_p is the diameter of the grain of the porous media, and here we assume it to be the half of the height of crops as the sixth assumption in this paper. K is the empirical value, ranging from 3 to 5, set as 4 here.

As a consequence, the porosity and permeability of the porous media are 0.5 and 0.0078125 m^2 by calculation, when the crops occupies a space of 0.9*0.8*0.5 in question two.

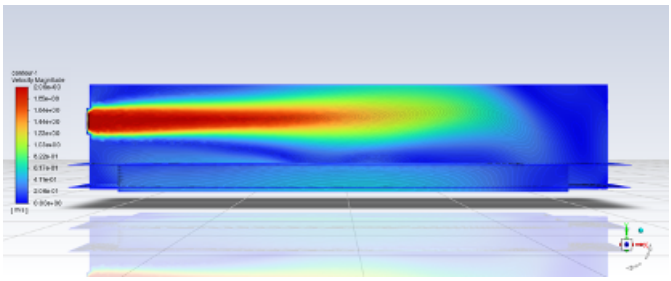
Thus, the longitudinal results given by ANSYS Fluent, showing the variation of temperature and velocity in the direction of the length of the greenhouse was given by Fig. (11), and distribution at the height of 0.5 meters and 0.1 meters simulated by ANSYS were exhibited in Fig. (12) and Fig. (13).

To assess the simulation accuracy of our mathematical model, we kept using the indicator R to show overlapping between the temperature and wind speed distribution intervals of the two models at the same location. In the height of 0.5m, the magnitudes of mathematical model were 31.2-37.6°C and 0-0.64m/s, whereas that of ANSYS were 31.85-37.85°C, and 0-0.61m/s. So, R achieved 89.84% and 95.31% for temperature and velocity simulation respectively. In the height of 0.1m, the magnitudes of mathematical model were 29-36°C and 0-0.8m/s, whereas that of ANSYS were 29.85-36.85°C, and 0-0.81m/s, with R reaching 87.86% and 98.77% individually.

In summary, the scaled model achieves better simulation of the micro-climatic conditions in the porous media region, and the two models showed great consistency,

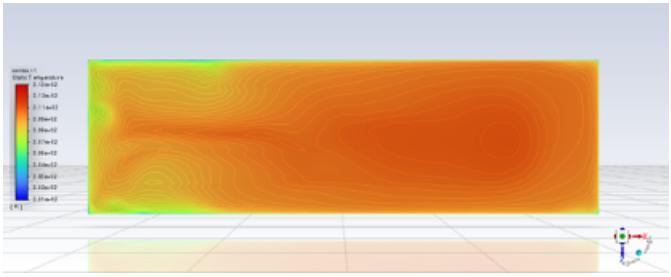


(a) Temperature fields

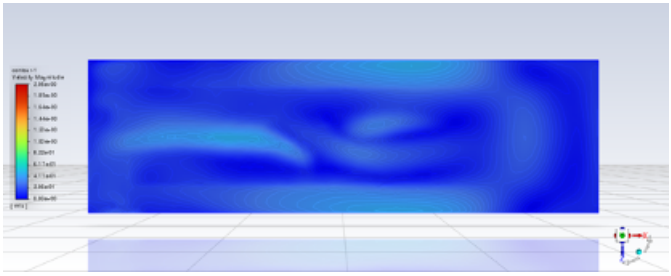


(b) Velocity fields

Figure 11 Distribution of target fields in the longitudual direction simulated by ANSYS



(a) Temperature fields



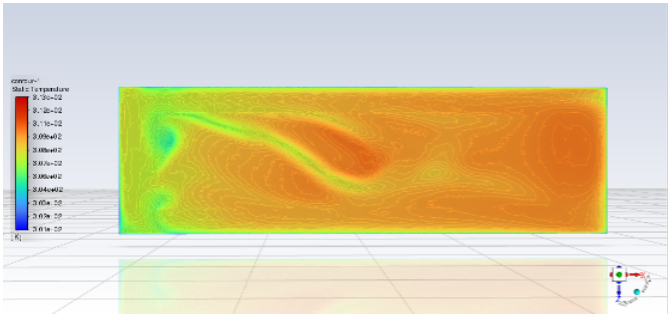
(b) Velocity fields

Figure 12 Distribution of target fields at the height of 0.5 meters simulated by ANSYS

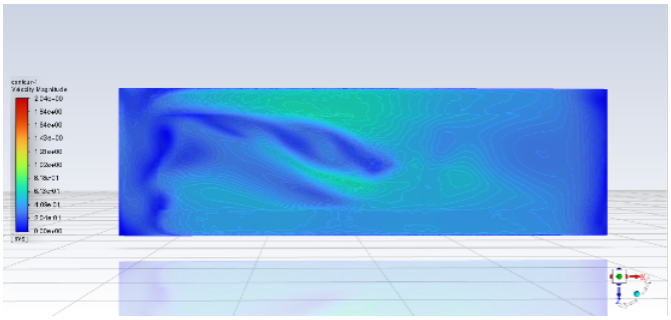
especially in simulating the environmental conditions near crop canopy and roots.

4.2.2 *Analyses*

From the temperature fields in Fig. (9), the comfortable zone with temperature ranging from 23°C to 26°C mainly exists in the length from 2m to 6m at the canopy of



(a) Temperature fields



(b) Velocity fields

Figure 13 Distribution of target fields at the height of 0.1 meters simulated by ANSYS

crops, while in roots, it mainly appears in the length from 1m to 3m. As for velocity, the comfortable velocity at the canopy appears in the length more than 4m, while in roots it exists in areas farther than 2m in the length direction.

In summary, we could conclude that crops in this greenhouse with the fans at the height of 1.3m, blowing hot winds at the temperature of 40 $°C$, actually results in an uncomfortable living condition for both canopy and roots of crops, because their requirements of temperature and wind velocity could be satisfied.

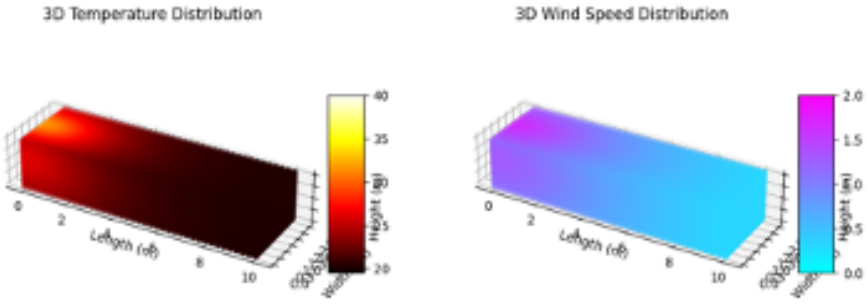
4.3 Question three

4.3.1 *Results and comparisons*

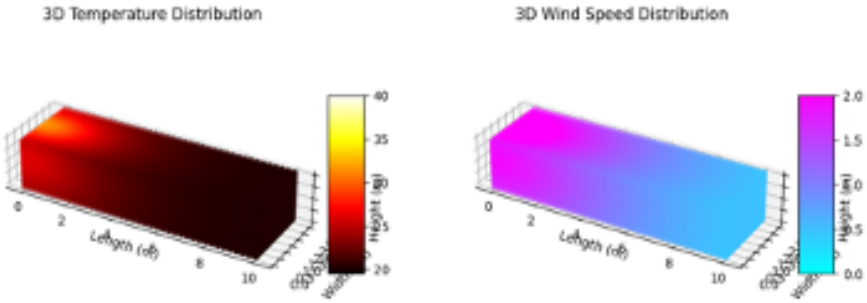
Based on the basic models build in question one and crop models established in question two, here we individually adjusted the velocity of warm air outlet from 2m/s to 3m/s, and lower the position of the greenhouse fan by moving it from 1.3 m to 1 m (see Appendix. 3)to make comparison with results in question two.

In scenario one, according to our mathematical model, when fans blew in air at the speed of 3m/s at the height of 1.3m, comparison with results in question 2 were shown in Fig. (14), where we could see that by increasing inlet velocity, although heat

did not obviously passed farther, velocity near the fans increases dramatically, and near fans areas receive more winds than ever.



(a) 2m/s in question two



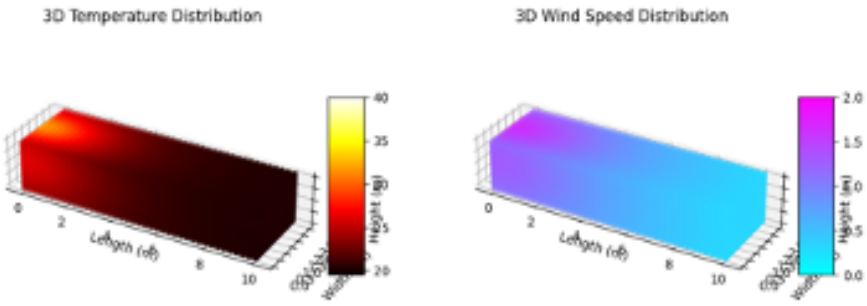
(b) 3m/s in question three

Figure 14 Three-dimensional results with different inlet velocity

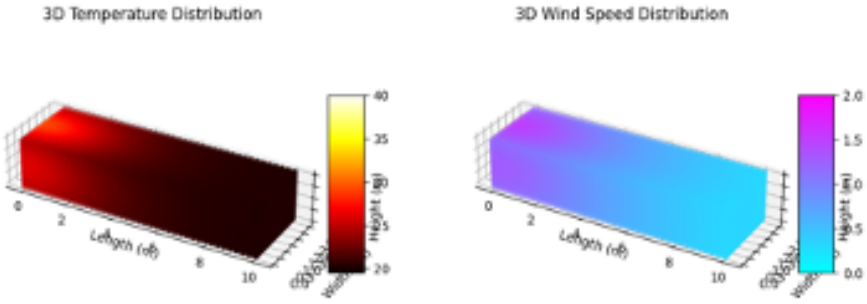
In scenario two, according to our mathematical model, when fans blew in air at the original speed of 2m/s at a lower height of 1m, comparison with results in question 2 were shown in Fig. (15), where in the height direction, we could observe a more uniform distribution, not only in temperature but also in velocity.

Meanwhile, at the canopy and roots of the crops, we could also observe from Fig. (16) and Fig. (17) that increasing wind velocity benefit canopy more than roots, because more areas in the height of 0.5m receive heat from hot wind, while roots receive a little. However, lower position of the fans could help roots more than canopy, because more areas in the height of 0.1m gets warmer and better uniformity.

In summary, from our observation, lower the position of the fans from 1.3m to 1m could achieve a more uniform distribution of fields, while increasing the velocity of air inlet make little progress in blowing wind deeper, but impairs the living condition of crops near the air inlet. In addition, increasing velocity in canopy achieves warmer living conditions in the canopy, and lower position of the fans benefits roots from better uniformity.

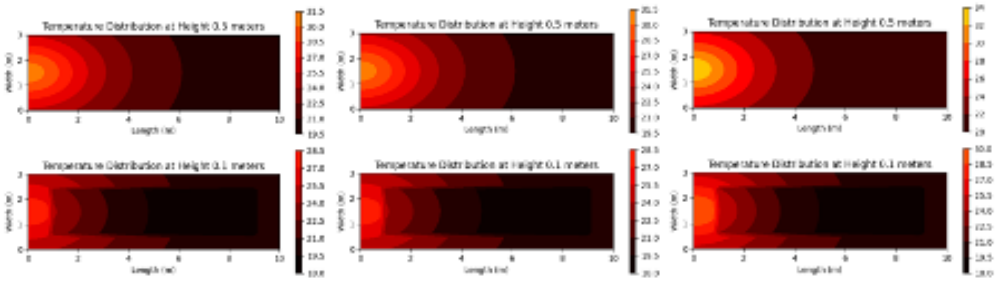


(a) 2m/s in question two



(b) 2m/s in question three

Figure 15 Three-dimensional results with different inlet position

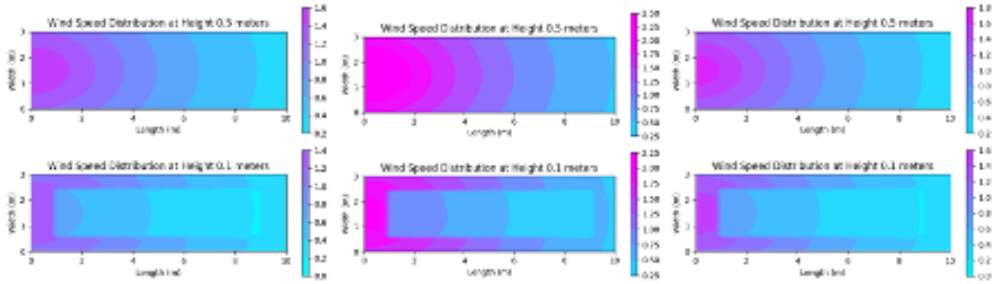


(a) question two (b) 3m/s in question three (c) 1m in question three

Figure 16 Temperature fields in different scenarios

4.3.2 *Validations*

Individually adjusting the velocity and position of the fans, we obtained target fields and made comparisons first with different scenarios and height in the area of temperature distribution in Fig. (18) and Fig. (19). It could obviously be observed that compared to increasing velocity in the height of 0.1m, lower position of the fans achieves better temperature distribution. More heat was transformed to deeper side in length direction. Similarly, compared to lower position, increasing velocity at canopy better uniforms the



(a) question two (b) 3m/s in question three (c) 1m in question three

Figure 17 Velocity fields in different scenarios

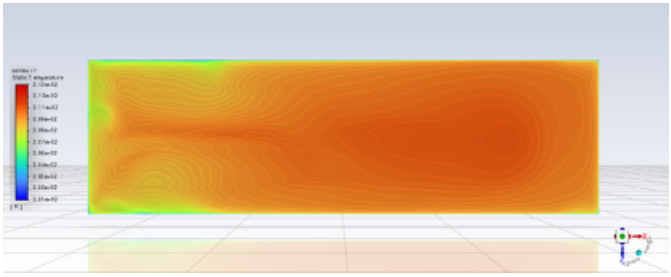
temperature distribution in the whole cuboid, which could also be concluded from the velocity field.

The consistent conclusion between mathematical model and ANSYS simulation could be quantified by our indicator R. In the height of 0.5m, the magnitudes of mathematical model were 32-38.4°C and 0-1.2m/s, whereas that of ANSYS were 32.85-38.85°C, and 0-1.22m/s. So, R achieved 86.72% and 98.36% for temperature and velocity simulation respectively. In the height of 0.1m, the magnitudes of mathematical model were 28.5-37.5°C and 0-1.05m/s, whereas that of ANSYS were 28.85-37.85°C, and 0-1m/s, with R reached 96.11% and 95.24% individually. The relatively high results, compared to the well-known ANSYS simulation demonstrate the reliability of our mathematical model and the deduction we made about the effects of adjustments.

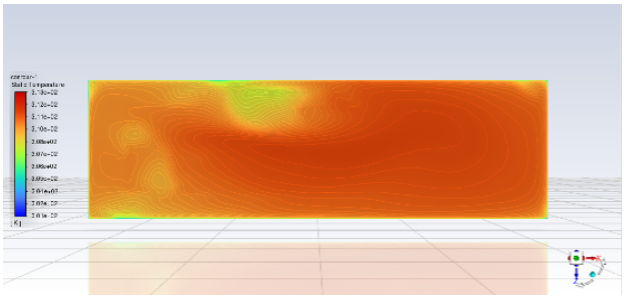
Overall, by applying the indicator, we showed that although the prediction results of the two models differ in some regions, they still show good agreement in most areas. This finding highlights the applicability and flexibility of the our model in simulating changes in complex environments. The reference provided by ANSYS simulation helped us in adjusting mathematical model to be reliable in further optimization, and validated our deduction on the direction of optimization, which would be beneficial in question four.

4.4 Question 4

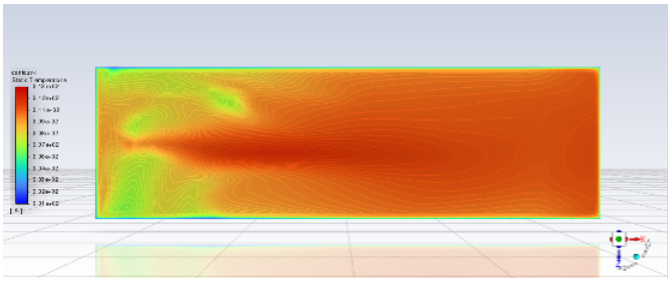
Based on the deficiency above, when it comes to optimizing design of glass greenhouse, among many of them, two main factors could be immediately came up with, the selection of crops and the optimization of the amount, location, speed and inlet temperature of fans.



(a) question two



(b) 3m/s in question three



(c) 1m in question three

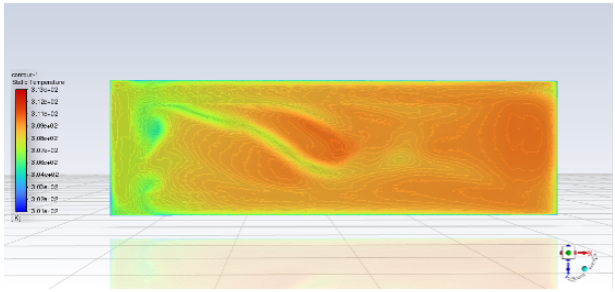
Figure 18 Temperature fields in different scenarios at the height of 0.5m

4.4.1 *Select Crops*

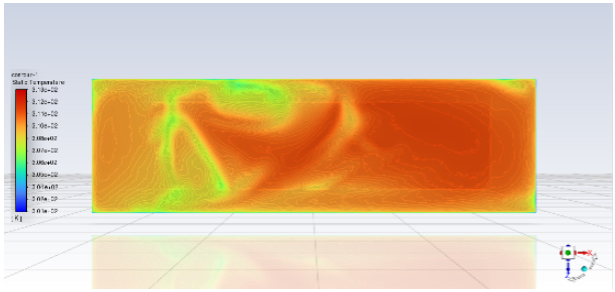
Our goal is to select crops that could fit high temperature and low velocity. According to our simulation results, for the first three conditions:

- For situations in question one, it's better to select crops that can endure temperatures within 31.2-37.6 $°C$, velocity below 0.48m/s at the height of 0.5m.
- For situations in question two, it's better to select crops that can endure temperatures within 31.2-37.6 $°C$, velocity below 0.61m/s at the height of 0.5m, and temperatures within 29-36 $°C$, velocity below 0.8m/s at the height of 0.1m.
- For situations in question three, it's better to select crops that can endure temperatures within 32-38.4 $°C$, velocity below 1.2m/s at the height of 0.5m, and temperatures within 28.5-37.5 $°C$, velocity below 1.05m/s at the height of 0.1m.

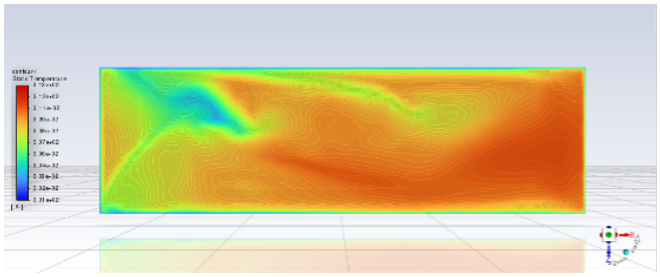
]



(a) question two



(b) 3m/s in question three



(c) 1m in question three

Figure 19 Temperature fields in different scenarios at the height of 0.1m

4.4.2 *Fans optimization by the Genetic Algorithm*

Here, we utilized a more complex but flexible algorithm, Genetic Algorithm (GA), with steps as follows.

(1) Objective function. Minimize the deviation between current conditions and the ideal environmental condition (23-26$°C$ and 0.3-1m/s)

(2) Constraints.

- The number of fans is the integer is **below five**.
- Fans could only be equipped on the walls or the roof. The value of x,y,z should either 0 or the size of the greehouse.

(3) Strategy. Each individual represents a full condition, including the number of fans, spatial coordinates, inlet velocity and temperature.

(4) Mutation. Randomly produce new variants.
(5) Evaluation and Selection. Make sure the survival of the fittest.
(6) Initialization. Initialize with a population of 1,000 individuals and iterate 50 times.

4.4.2.1 Innovative points

(1) Length magnification. Considering the direction and magnitude of the fans, we devise a factor, *length magnification*, to strength its power in the specific direction of the fans' head, other than the other vertical directions.
(2) Convergence of GA. We regarded each generation in the GA algorithm as a state in the Markov Chain, which meets the criteria of the dependence to current generation and the independence to the last generation. As a consequence, in accord with the Markov Chain Convergence Theorem, we will finally achieve convergence of GA, under the implementation of Markov Chain.

4.4.2.2 Results

The flexible GA algorithm and the convergence guaranteed by the Markov Chain provided us with flexible solutions to the optimization of fans (see Appendix. (4)). Under the constraints set before, here, we displayed two solutions that both satisfied the temperature and velocity requirement, based on our mathematical models built before.

The first scheme contained five fans, with information provided in Table. (2). The location is clearly shown in Fig. (20. We could validate the feasibility by exhibiting target fields in Fig. (21) and Fig. (22). Whereas, **The second scheme** contained three fans, with information provided in Table. (3). Fig. (23 directly exhibited the locations of fans. Also, we could validate the feasibility by target fields in Fig. (24).

V. Pros and Cons

5.1 Pros

Our models are able to provide an in-depth spatial understanding of the climate conditions within the greenhouse, especially the spatial distribution of temperature and wind speed. We are able to accurately predict and evaluate the microclimate conditions in the greenhouse planting crop growth. Through Python programming and the application

Table 2 Configurations of five-fans scheme

Speed(m/s)	Temperature(°C)	x(m)	y(m)	z(m)
0.72603	26.08	2.87667	0	0.00068
0.79941	26.31	0	0.20255	1.05669
0.73218	26.70	2.38155	0	0.68965
0.78105	26.16	1.99720	1.17448	2
0.79096	26.49	3.74524	0	2

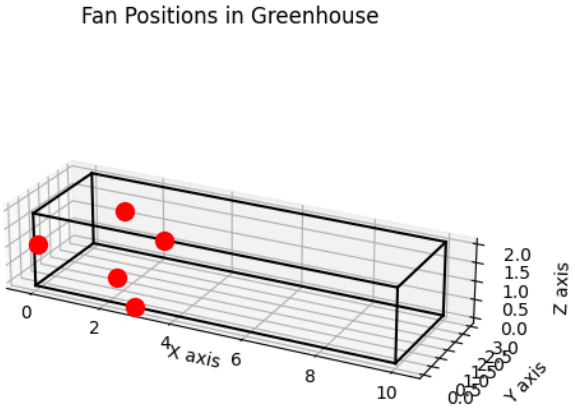


Figure 20 Locations of the five fans

of CFD software, ANSYS Fluent, we are able to represent complex three-dimensional temperature and wind speed distribution, providing visual aids for greenhouse design and optimization of crop growing conditions, at specific heights (e.g., 0.5 m and 0.1 m). In addition, the application of genetic algorithms in optimizing greenhouse fan designs has demonstrated a high degree of flexibility and efficiency. This approach takes into account not only the number and location of fans, but also factors such as wind speed and temperature, to generate a diverse range of solutions that provide an optimal environment for different crops and different greenhouse conditions. Finally, the application of these models has a certain degree of universality and scalability that may contribute to planning phase of greenhouse design and in the adaptation of existing greenhouse environments.

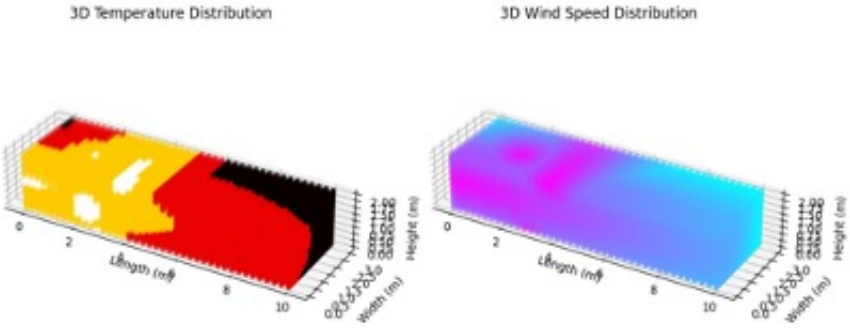


Figure 21 Three dimensional fields of the five-fans scheme

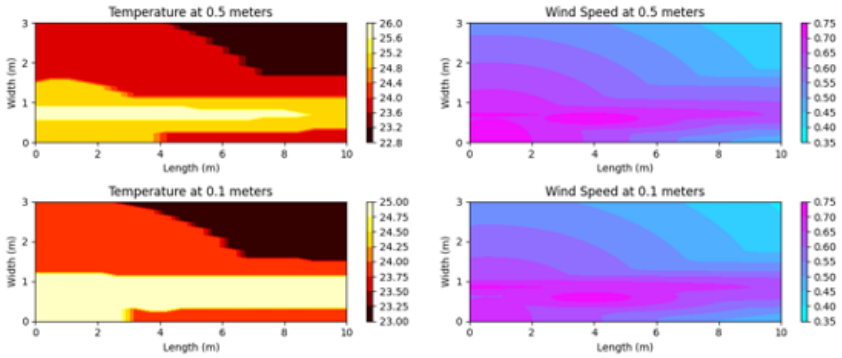


Figure 22 Target fields at two cross-sections of the five-fans scheme

5.2 Cons

Despite achievements, some deficiency should not be ignored. Firstly, the accuracy of the model relies heavily on the veracity of the initial assumptions and input parameters. Secondly, simple assumptions about porous media in crop models, may affect the accuracy of our results. Thirdly, since the steady-state case is mainly considered in the modeling, the energy conservation equation is simplified to a certain extent. Besides, when considering the resistance of the canopy and roots of crops to wind, only the effect of distance from the fan on energy transfer was considered, and no more detailed distinction was made between leaves and roots. Next, although genetic algorithm provides an effective optimization method, it is still a trial-and-error method, and its convergence and the quality of the final solution depend to some extent on the selection of algorithm parameters and the diversity of populations. Finally, the capability of the model may be limited by the specific application, e.g., external environmental factors such as solar radiation, external air currents, etc.

Table 3 Configurations of five-fans scheme

Speed(m/s)	Temperature(°C)	x(m)	y(m)	z(m)
0.73980	26.47	2.06455	3	0.22216
0.77190	26.24	0	0.16405	0.86741
0.65997	26.49	6.07402	2.80944	2

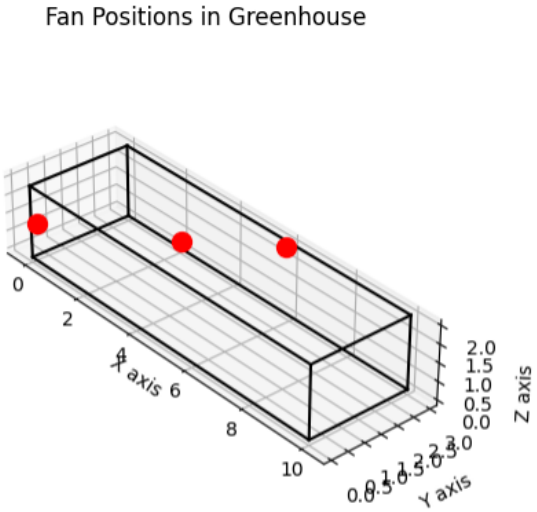
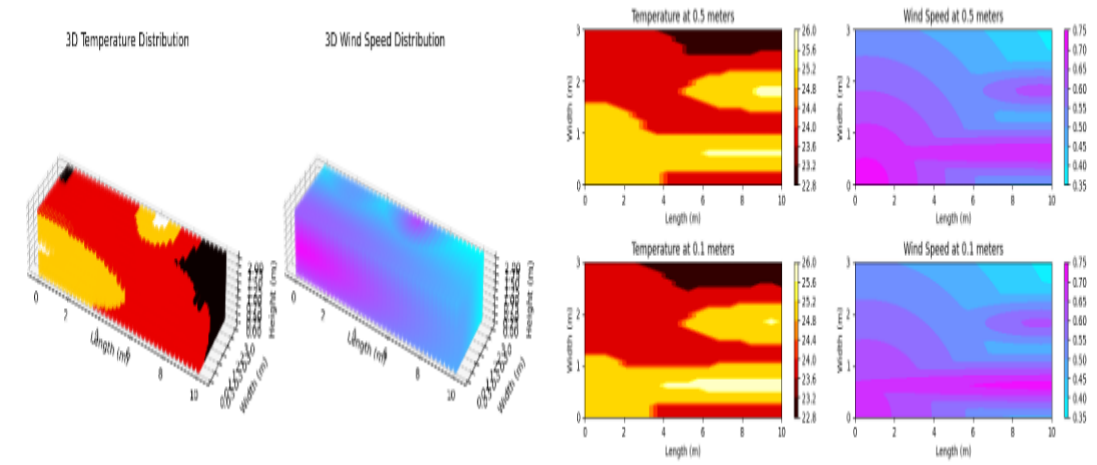


Figure 23 Locations of the three fans



(a) Three dimensional fields (b) Target fields at two cross-sections

Figure 24 The three-fans scheme

VI. References

[1] Stanghellini, C., Kubota, C. Greenhouse Technology and Management. CRC Press. 2010.

[2] Guo, L., et al. CFD simulation of airflow and temperature distribution in a greenhouse with a heat pump heating system. Biosystems Engineering, 183, 44-57. 2019.

VII. Appendix

Listing 1: The Python Source code of question one

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parameter Settings
length, width, height = 10, 3, 2 # Greenhouse dimensions (m)
fan_speed = 2 # Fan speed (m/s)
fan_temp = 40 # Fan temperature
ambient_temp = 20 # Initial and external temperature

# Create grid
x = np.linspace(0, length, 40)
y = np.linspace(0, width, 20)
z = np.linspace(0, height, 20)
X, Y, Z = np.meshgrid(x, y, z)

# Fan position parameters
fan_position = (0, 1.5, 1.3) # Assume the fan is located on the left
    side of the greenhouse, center height at 1.3m

# Decay control parameters
temp_decay_control = 0.2 # Controls the degree of temperature decay
wind_decay_control = 1.9 # Controls the degree of wind speed decay

# Amplification factor
length_magnification = 0.2

# Calculate the distance from each point to the fan
def distance_to_fan(X, Y, Z, fan_position, length_magnification):
    fan_x, fan_y, fan_z = fan_position
    return np.sqrt((length_magnification * (X - fan_x))**2 + (Y -
        fan_y)**2 + (Z - fan_z)**2)
```

```
# Update temperature and wind speed calculation functions with the new
    distance calculation method
def calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return ambient_temp + (fan_temp - ambient_temp) *
        np.exp(-decay_control * distance)

temperature = calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, temp_decay_control, length_magnification)

def calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return fan_speed * np.exp(-decay_control * distance)

wind_speed = calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    wind_decay_control, length_magnification)

# Create 3D plots for temperature and wind speed distribution
fig = plt.figure(figsize=(12, 5))

# Plot 3D temperature distribution
ax1 = fig.add_subplot(121, projection='3d')
# Set the view angle
# ax1.view_init(elev=20, azim=-180)
# ax1.view_init(elev=20, azim=20)
# Set the x-axis and y-axis to use the same scale
ax1.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
temp_plot = ax1.scatter(X, Y, Z, c=temperature, cmap='hot',vmin=19.5,
    vmax=40)
ax1.set_title('3D Temperature Distribution')
ax1.set_xlabel('Length (m)')
ax1.set_ylabel('Width (m)')
ax1.set_zlabel('Height (m)')
```

```
fig.colorbar(temp_plot, ax=ax1, shrink=0.5, aspect=5)

# Plot wind speed distribution
ax2 = fig.add_subplot(122, projection='3d')
# Set the view angle
# ax2.view_init(elev=20, azim=-180)
# ax2.view_init(elev=20, azim=20)

# Set the x-axis and y-axis to use the same scale
ax2.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
wind_plot = ax2.scatter(X, Y, Z, c=wind_speed, cmap='cool',vmin=0,
    vmax=2.0)
ax2.set_title('3D Wind Speed Distribution')
ax2.set_xlabel('Length (m)')
ax2.set_ylabel('Width (m)')
ax2.set_zlabel('Height (m)')
fig.colorbar(wind_plot, ax=ax2, shrink=0.5, aspect=5)

# Calculate temperature and wind speed distribution at a height of 0.5
    meters
half_height_index = np.argmin(np.abs(z - 0.5)) # Index corresponding to
    the height of 0.5 meters
temperature_half_height = temperature[:, :, half_height_index]
wind_speed_half_height = wind_speed[:, :, half_height_index]

# Plot temperature and wind speed distribution at a height of 0.5 meters
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Temperature Distribution at Height 0.5 meters
c1 = ax1.contourf(x, y, temperature_half_height, cmap='hot', vmin=19.5,
    vmax=40)
fig.colorbar(c1, ax=ax1)
ax1.set_aspect('equal')
ax1.set_title('Temperature Distribution at Height 0.5 meters')
ax1.set_xlabel('Length (m)')
ax1.set_ylabel('Width (m)')
```

```
# Wind Speed Distribution at Height 0.5 meters
c2 = ax2.contourf(x, y, wind_speed_half_height, cmap='cool',vmin=0,
    vmax=2.0)
fig.colorbar(c2, ax=ax2)
ax2.set_aspect('equal')
ax2.set_title('Wind Speed Distribution at Height 0.5 meters')
ax2.set_xlabel('Length (m)')
ax2.set_ylabel('Width (m)')

plt.show()
```

Listing 2: The Python source code for question two

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parameter Settings
length, width, height = 10, 3, 2 # Greenhouse dimensions (m)
fan_speed = 2 # Fan speed (m/s)
fan_temp = 40 # Fan temperature (degree)
ambient_temp = 20 # Initial and external temperature (degree)

# Create grid
x = np.linspace(0, length, 40)
y = np.linspace(0, width, 20)
z = np.linspace(0, height, 20)
X, Y, Z = np.meshgrid(x, y, z)

# Fan position parameters
fan_position = (0, 1.5, 1.3) # Assume the fan is located on the left
    side of the greenhouse, center height at 1.3m

# Decay control parameters
temp_decay_control = 0.8 # Controls the degree of temperature decay
wind_decay_control = 0.3 # Controls the degree of wind speed decay

# Amplification factor
```

```
length_magnification = 0.6

# Calculate the distance from each point to the fan
def distance_to_fan(X, Y, Z, fan_position, length_magnification):
    fan_x, fan_y, fan_z = fan_position
    return np.sqrt((length_magnification * (X - fan_x))**2 + (Y -
        fan_y)**2 + (Z - fan_z)**2)

# Update temperature and wind speed calculation functions with the new
    distance calculation method
def calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return ambient_temp + (fan_temp - ambient_temp) *
        np.exp(-decay_control * distance)

temperature = calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, temp_decay_control, length_magnification)

def calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return fan_speed * np.exp(-decay_control * distance)

wind_speed = calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    wind_decay_control, length_magnification)

# Define the porous medium region
porous_start, porous_end = 1, 9 # Start and end positions of the porous
    medium in the length direction
porous_width_start, porous_width_end = 0.5, 2.5 # Start and end
    positions in the width direction
porous_height = 0.5 # Height of the porous medium

# Check if a point is within the porous medium region
```

```
def is_inside_porous(X, Y, Z):
    return ((porous_start <= X) & (X <= porous_end) &
            (porous_width_start <= Y) & (Y <= porous_width_end) &
            (Z <= porous_height))

# Update temperature and wind speed calculations to include the
    influence of the porous medium
def calculate_temperature_with_porous(X, Y, Z, fan_temp, ambient_temp,
    fan_position, decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    temperature = ambient_temp + (fan_temp - ambient_temp) *
        np.exp(-decay_control * distance)
    # Reduce temperature within the porous medium region
    temperature[is_inside_porous(X, Y, Z)] *= 0.9
    return temperature

def calculate_wind_speed_with_porous(X, Y, Z, fan_speed, fan_position,
    decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    wind_speed = fan_speed * np.exp(-decay_control * distance)
    # Significantly reduce wind speed within the porous medium region
    wind_speed[is_inside_porous(X, Y, Z)] *= 0.5
    return wind_speed

# Use new functions to calculate temperature and wind speed
temperature = calculate_temperature_with_porous(X, Y, Z, fan_temp,
    ambient_temp, fan_position, temp_decay_control,
    length_magnification)
wind_speed = calculate_wind_speed_with_porous(X, Y, Z, fan_speed,
    fan_position, wind_decay_control, length_magnification)

# Create 3D plots for temperature and wind speed distribution
fig = plt.figure(figsize=(12, 5))

# Plot 3D temperature distribution
```

```
ax1 = fig.add_subplot(121, projection='3d')
# Set the view angle
# ax1.view_init(elev=20, azim=-180)
# ax1.view_init(elev=20, azim=20)
# Set the x-axis and y-axis to use the same scale
ax1.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
temp_plot = ax1.scatter(X, Y, Z, c=temperature, cmap='hot',vmin=19.5,
    vmax=40)
ax1.set_title('3D Temperature Distribution')
ax1.set_xlabel('Length (m)')
ax1.set_ylabel('Width (m)')
ax1.set_zlabel('Height (m)')
fig.colorbar(temp_plot, ax=ax1, shrink=0.5, aspect=5)

# Plot 3D wind speed distribution
ax2 = fig.add_subplot(122, projection='3d')
# Set the view angle
# ax2.view_init(elev=20, azim=-180)
# ax2.view_init(elev=20, azim=20)

# Set the x-axis and y-axis to use the same scale
ax2.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
wind_plot = ax2.scatter(X, Y, Z, c=wind_speed, cmap='cool',vmin=0,
    vmax=2.0)
ax2.set_title('3D Wind Speed Distribution')
ax2.set_xlabel('Length (m)')
ax2.set_ylabel('Width (m)')
ax2.set_zlabel('Height (m)')
fig.colorbar(wind_plot, ax=ax2, shrink=0.5, aspect=5)

# Calculate temperature and wind speed distribution at a height of 0.5
    meters
half_height_index = np.argmin(np.abs(z - 0.5)) # Index corresponding to
    the height of 0.5 meters
temperature_half_height = temperature[:, :, half_height_index]
wind_speed_half_height = wind_speed[:, :, half_height_index]
```

```
# Calculate temperature and wind speed distribution at a height of 0.1
    meters
low_height_index = np.argmin(np.abs(z - 0.1)) # Index corresponding to
    the height of 0.1 meters
temperature_low_height = temperature[:, :, low_height_index]
wind_speed_low_height = wind_speed[:, :, low_height_index]


# Plot temperature and wind speed distribution at a height of 0.5
    meters and at 0.1 meters
fig, axs = plt.subplots(2, 2, figsize=(12, 5))

# Temperature Distribution at Height 0.5 meters
c1 = axs[0, 0].contourf(x, y, temperature_half_height, cmap='hot',
    vmin=19.5, vmax=40)
fig.colorbar(c1, ax=axs[0, 0])
axs[0, 0].set_aspect('equal')
axs[0, 0].set_title('Temperature Distribution at Height 0.5 meters')
axs[0, 0].set_xlabel('Length (m)')
axs[0, 0].set_ylabel('Width (m)')

# Wind Speed Distribution at Height 0.5 meters
c2 = axs[0, 1].contourf(x, y, wind_speed_half_height, cmap='cool',
    vmin=0, vmax=2.0)
fig.colorbar(c2, ax=axs[0, 1])
axs[0, 1].set_aspect('equal')
axs[0, 1].set_title('Wind Speed Distribution at Height 0.5 meters')
axs[0, 1].set_xlabel('Length (m)')
axs[0, 1].set_ylabel('Width (m)')

# Temperature Distribution at Height 0.1 meters
c3 = axs[1, 0].contourf(x, y, temperature_low_height, cmap='hot',
    vmin=19.5, vmax=40)
fig.colorbar(c3, ax=axs[1, 0])
axs[1, 0].set_aspect('equal')
axs[1, 0].set_title('Temperature Distribution at Height 0.1 meters')
axs[1, 0].set_xlabel('Length (m)')
```

```
axs[1, 0].set_ylabel('Width (m)')

# Wind Speed Distribution at Height 0.1 meters
c4 = axs[1, 1].contourf(x, y, wind_speed_low_height, cmap='cool',
    vmin=0, vmax=2.0)
fig.colorbar(c4, ax=axs[1, 1])
axs[1, 1].set_aspect('equal')
axs[1, 1].set_title('Wind Speed Distribution at Height 0.1 meters')
axs[1, 1].set_xlabel('Length (m)')
axs[1, 1].set_ylabel('Width (m)')

plt.tight_layout()
plt.show()
```

Listing 3: The Python source code for question three

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Parameter Settings
length, width, height = 10, 3, 2 # Greenhouse dimensions (m)
fan_speed = 2 # Fan speed (m/s)
fan_temp = 40 # Fan temperature (C)
ambient_temp = 20 # Initial and external temperature (C)

# Create grid
x = np.linspace(0, length, 40)
y = np.linspace(0, width, 20)
z = np.linspace(0, height, 20)
X, Y, Z = np.meshgrid(x, y, z)

# Fan position parameters
fan_position = (0, 1.5, 1.3) # Assume the fan is located on the left
    side of the greenhouse, center height at 1.3m

# Decay control parameters
temp_decay_control = 0.2 # Controls the degree of temperature decay
```

```
wind_decay_control = 2 # Controls the degree of wind speed decay

# Amplification factor
length_magnification = 0.2

# Calculate the distance from each point to the fan
def distance_to_fan(X, Y, Z, fan_position, length_magnification):
    fan_x, fan_y, fan_z = fan_position
    return np.sqrt((length_magnification * (X - fan_x))**2 + (Y -
        fan_y)**2 + (Z - fan_z)**2)

# Update temperature and wind speed calculation functions with the new
    distance calculation method
def calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return ambient_temp + (fan_temp - ambient_temp) *
        np.exp(-decay_control * distance)

temperature = calculate_temperature(X, Y, Z, fan_temp, ambient_temp,
    fan_position, temp_decay_control, length_magnification)

def calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    return fan_speed * np.exp(-decay_control * distance)

wind_speed = calculate_wind_speed(X, Y, Z, fan_speed, fan_position,
    wind_decay_control, length_magnification)

# Define the porous medium region
porous_start, porous_end = 1, 9 # Start and end positions of the porous
    medium in the length direction
porous_width_start, porous_width_end = 0.5, 2.5 # Start and end
    positions in the width direction
```

```
porous_height = 0.5 # Height of the porous medium

# Check if a point is within the porous medium region
def is_inside_porous(X, Y, Z):
    return ((porous_start <= X) & (X <= porous_end) &
            (porous_width_start <= Y) & (Y <= porous_width_end) &
            (Z <= porous_height))

# Update temperature and wind speed calculations to include the
    influence of the porous medium
def calculate_temperature_with_porous(X, Y, Z, fan_temp, ambient_temp,
    fan_position, decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    temperature = ambient_temp + (fan_temp - ambient_temp) *
        np.exp(-decay_control * distance)
   # Reduce temperature within the porous medium region
    temperature[is_inside_porous(X, Y, Z)] *= 0.87
    return temperature

def calculate_wind_speed_with_porous(X, Y, Z, fan_speed, fan_position,
    decay_control, length_magnification):
    distance = distance_to_fan(X, Y, Z, fan_position,
        length_magnification)
    wind_speed = fan_speed * np.exp(-decay_control * distance)
    # Significantly reduce wind speed within the porous medium region
    wind_speed[is_inside_porous(X, Y, Z)] *= 2
    return wind_speed

# Use new functions to calculate temperature and wind speed
temperature = calculate_temperature_with_porous(X, Y, Z, fan_temp,
    ambient_temp, fan_position, temp_decay_control,
    length_magnification)
wind_speed = calculate_wind_speed_with_porous(X, Y, Z, fan_speed,
    fan_position, wind_decay_control, length_magnification)
```

```
# Create 3D plots for temperature and wind speed distribution
fig = plt.figure(figsize=(12, 5))
# Plot 3D temperature distribution
ax1 = fig.add_subplot(121, projection='3d')
# Set the view angle
# ax1.view_init(elev=20, azim=-180)
# ax1.view_init(elev=20, azim=20)

# Set the x-axis and y-axis to use the same scale
ax1.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
temp_plot = ax1.scatter(X, Y, Z, c=temperature, cmap='hot',vmin=19.5,
    vmax=40)
ax1.set_title('3D Temperature Distribution')
ax1.set_xlabel('Length (m)')
ax1.set_ylabel('Width (m)')
ax1.set_zlabel('Height (m)')
fig.colorbar(temp_plot, ax=ax1, shrink=0.5, aspect=5)

# Plot wind speed distribution
ax2 = fig.add_subplot(122, projection='3d')
# Set the view angle
# ax2.view_init(elev=20, azim=-180)
# ax2.view_init(elev=20, azim=20)

# Set the x-axis and y-axis to use the same scale
ax2.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
wind_plot = ax2.scatter(X, Y, Z, c=wind_speed, cmap='cool',vmin=0,
    vmax=2.0)
ax2.set_title('3D Wind Speed Distribution')
ax2.set_xlabel('Length (m)')
ax2.set_ylabel('Width (m)')
ax2.set_zlabel('Height (m)')
fig.colorbar(wind_plot, ax=ax2, shrink=0.5, aspect=5)

# Calculate temperature and wind speed distribution at a height of 0.5
    meters
```

```
half_height_index = np.argmin(np.abs(z - 0.5)) # Index corresponding to
    the height of 0.5 meters
temperature_half_height = temperature[:, :, half_height_index]
wind_speed_half_height = wind_speed[:, :, half_height_index]

# Calculate temperature and wind speed distribution at a height of 0.1
    meters
low_height_index = np.argmin(np.abs(z - 0.1)) # Index corresponding to
    the height of 0.1 meters
temperature_low_height = temperature[:, :, low_height_index]
wind_speed_low_height = wind_speed[:, :, low_height_index]



# Plot temperature and wind speed distribution at a height of 0.5
    meters and at 0.1 meters
fig, axs = plt.subplots(2, 2, figsize=(12, 5))

# Temperature Distribution at Height 0.5 meters
c1 = axs[0, 0].contourf(x, y, temperature_half_height, cmap='hot',
    vmin=19.5, vmax=40)
fig.colorbar(c1, ax=axs[0, 0])
axs[0, 0].set_aspect('equal')
axs[0, 0].set_title('Temperature Distribution at Height 0.5 meters')
axs[0, 0].set_xlabel('Length (m)')
axs[0, 0].set_ylabel('Width (m)')

# Wind Speed Distribution at Height 0.5 meters
c2 = axs[0, 1].contourf(x, y, wind_speed_half_height, cmap='cool',
    vmin=0, vmax=2.0)
fig.colorbar(c2, ax=axs[0, 1])
axs[0, 1].set_aspect('equal')
axs[0, 1].set_title('Wind Speed Distribution at Height 0.5 meters')
axs[0, 1].set_xlabel('Length (m)')
axs[0, 1].set_ylabel('Width (m)')

# Temperature Distribution at Height 0.1 meters
c3 = axs[1, 0].contourf(x, y, temperature_low_height, cmap='hot',
```

```
    vmin=19.5, vmax=40)
fig.colorbar(c3, ax=axs[1, 0])
axs[1, 0].set_aspect('equal')
axs[1, 0].set_title('Temperature Distribution at Height 0.1 meters')
axs[1, 0].set_xlabel('Length (m)')
axs[1, 0].set_ylabel('Width (m)')

# Wind Speed Distribution at Height 0.1 meters
c4 = axs[1, 1].contourf(x, y, wind_speed_low_height, cmap='cool',
    vmin=0, vmax=2.0)
fig.colorbar(c4, ax=axs[1, 1])
axs[1, 1].set_aspect('equal')
axs[1, 1].set_title('Wind Speed Distribution at Height 0.1 meters')
axs[1, 1].set_xlabel('Length (m)')
axs[1, 1].set_ylabel('Width (m)')

plt.tight_layout()
plt.show()
```

Listing 4: The Python source code for question four

```
import numpy as np
from deap import base, creator, tools, algorithms
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Greenhouse Dimensions and Environmental Parameters
length, width, height = 10, 3, 2
ambient_temp = 20

# Genetic Algorithm Parameters
population_size = 1000
number_of_generations = 50
crossover_probability = 0.7
mutation_probability = 0.2

# Maximum Number of Fans
```

```
max_fans_allowed = 5

# Create Grid
x = np.linspace(0, length, 40)
y = np.linspace(0, width, 20)
z = np.linspace(0, height, 20)
X, Y, Z = np.meshgrid(x, y, z, indexing='ij')

# Decay control parameters
temp_decay_control = 0.2 # Controls the degree of temperature decay
wind_decay_control = 0.2 # Controls the degree of wind speed decay

# Amplification factor
length_magnification = 0.2

# Check if fan positions are around the greenhouse perimeter and top
def is_fan_position_valid(x, y, z):
    return (x == 0 or x == length or y == 0 or y == width or z == height)

def update_temperature_and_wind_speed(temperature_distribution,
    wind_speed_distribution, fan_speed, fan_temp, fan_x, fan_y, fan_z,
    decay_control_temp, decay_control_wind):
    for i in range(temperature_distribution.shape[0]):
        for j in range(temperature_distribution.shape[1]):
            for k in range(temperature_distribution.shape[2]):
                # Apply magnification factor based on fan positions
                dx, dy, dz = x[i] - fan_x, y[j] - fan_y, z[k] - fan_z

                if fan_x == 0: # Fans on the left wall
                    distance = np.sqrt((length_magnification * dx)**2 +
                        dy**2 + dz**2)
                elif fan_x == length: # Fans on the right wall
                    distance = np.sqrt((length_magnification * -dx)**2 +
                        dy**2 + dz**2)
                elif fan_y == 0: # Fans on the front wall
                    distance = np.sqrt(dx**2 + (length_magnification *
                        dy)**2 + dz**2)
```

```
            elif fan_y == width: # Fans on the back wall
                distance = np.sqrt(dx**2 + (length_magnification *
                    -dy)**2 + dz**2)
            elif fan_z == height: # Fans on the ceiling
                distance = np.sqrt(dx**2 + dy**2 +
                    (length_magnification * -dz)**2)

            # Calculate the influence of distance on temperature and
                wind speed
            temp_effect = fan_temp * np.exp(-decay_control_temp *
                distance)
            wind_effect = fan_speed * np.exp(-decay_control_wind *
                distance)

            # Update temperature and wind speed distribution
            temperature_distribution[i, j, k] =
                max(temperature_distribution[i, j, k], temp_effect +
                ambient_temp)
            wind_speed_distribution[i, j, k] =
                max(wind_speed_distribution[i, j, k], wind_effect)

    return temperature_distribution, wind_speed_distribution


# Objective function: Calculate the deviation of temperature and wind
    speed distribution inside the greenhouse
def evaluate(individual):
    num_fans = int(individual[0])
    fan_parameters = individual[1:]

    num_fans = abs(int(individual[0]))
    num_fans = min(num_fans, max_fans_allowed)

    # Initialize temperature and wind speed distribution
    temperature_distribution = np.full((length, width, height),
        ambient_temp)
    wind_speed_distribution = np.zeros((length, width, height))
```

```
    # Handle each fan
    for i in range(num_fans):
        fan_speed, fan_temp, fan_x, fan_y, fan_z = fan_parameters[i *
            5:(i + 1) * 5]

        # Check if the fan positions are valid
        if not is_fan_position_valid(fan_x, fan_y, fan_z):
            return 9999, # High penalty for violating position constraints

        # Update temperature and wind speed distribution
        temperature_distribution, wind_speed_distribution =
            update_temperature_and_wind_speed(
            temperature_distribution, wind_speed_distribution, fan_speed,
                fan_temp, fan_x, fan_y, fan_z, temp_decay_control,
                wind_decay_control)

    # Calculate deviation from ideal conditions (example calculation
        method)
    temp_error = np.mean((temperature_distribution - 24.5) ** 2)
    wind_error = np.mean((wind_speed_distribution - 0.65) ** 2)

    return temp_error + wind_error,

# Set up the DEAP framework
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("attr_num_fans", random.randint, 1, max_fans_allowed)
toolbox.register("attr_float", random.uniform, 0.5, 0.8) # wind speed
toolbox.register("attr_temp", random.uniform, 6, 8) # temperature
toolbox.register("attr_pos", random.uniform, 0, length) # location

toolbox.register("individual", tools.initCycle, creator.Individual,
                 (toolbox.attr_num_fans, toolbox.attr_float,
                     toolbox.attr_temp, toolbox.attr_pos,
```

```
                    toolbox.attr_pos, toolbox.attr_pos),
                    n=max_fans_allowed)
toolbox.register("population", tools.initRepeat, list,
    toolbox.individual)

toolbox.register("evaluate", evaluate)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=1, indpb=0.2)
toolbox.register("select", tools.selTournament, tournsize=3)

# Generate valid fan positions
def generate_fan_position():
    # Randomly choose which coordinate to set as 0 or the maximum value
        (ensuring on the wall)
    wall_pos = random.choice(['x', 'y', 'z'])
    max_values = {'x': length, 'y': width, 'z': height}

    if wall_pos == 'x':
        x_pos = random.choice([0, max_values['x']])
        return (x_pos, random.uniform(0, width), random.uniform(0,
            height))
    elif wall_pos == 'y':
        y_pos = random.choice([0, max_values['y']])
        return (random.uniform(0, length), y_pos, random.uniform(0,
            height))
    else: # 'z'
        # Ensure fans are not on the floor, i.e., the z-coordinate
            cannot be 0
        z_pos = height
        return (random.uniform(0, length), random.uniform(0, width),
            z_pos)

# Modify the way individuals are created to include new position
    constraints
def create_individual():
    individual = [random.randint(1, max_fans_allowed)] # numver
    for _ in range(max_fans_allowed):
```

```
        fan_speed = random.uniform(0.5, 0.8) # wind speed
        fan_temp = random.uniform(6, 8) # temperature
        fan_pos = generate_fan_position() # location
        individual.extend([fan_speed, fan_temp] + list(fan_pos))
    return creator.Individual(individual)

toolbox.register("individual", create_individual)
toolbox.register("population", tools.initRepeat, list,
    toolbox.individual)

# Custom mutation function to ensure fan positions adhere to new
    constraints
def mutate(individual):
    for i in range(1, len(individual), 5):
        if random.random() < mutation_probability:
            individual[i] = random.uniform(0.5, 0.8) # wind speed
            individual[i+1] = random.uniform(6, 8) # temperature
            individual[i+2], individual[i+3], individual[i+4] =
                generate_fan_position() # location
    return individual,

toolbox.register("mutate", mutate)

# Run the genetic algorithm
population = toolbox.population(n=population_size)
result = algorithms.eaSimple(population, toolbox,
    cxpb=crossover_probability, mutpb=mutation_probability,
                         ngen=number_of_generations, verbose=True)

# Function to plot the optimal fan configuration
def plot_fan_positions(best_individual):
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')

    # Plot the boundary of the greenhouse
    ax.plot([0, length], [0, 0], [0, 0], color='black') # bottom boundary
    ax.plot([0, length], [width, width], [0, 0], color='black')
```

```
    ax.plot([0, 0], [0, width], [0, 0], color='black')
    ax.plot([length, length], [0, width], [0, 0], color='black')

    ax.plot([0, length], [0, 0], [height, height], color='black') # top
        boundary
    ax.plot([0, length], [width, width], [height, height], color='black')
    ax.plot([0, 0], [0, width], [height, height], color='black')
    ax.plot([length, length], [0, width], [height, height],
        color='black')

    ax.plot([0, 0], [0, 0], [0, height], color='black') # pillar
    ax.plot([length, length], [0, 0], [0, height], color='black')
    ax.plot([0, 0], [width, width], [0, height], color='black')
    ax.plot([length, length], [width, width], [0, height], color='black')

    num_fans = int(best_individual[0])
    fan_parameters = best_individual[1:]

    # Plot the position of each fan
    for i in range(num_fans):
        fan_speed, fan_temp, fan_x, fan_y, fan_z = fan_parameters[i *
            5:(i + 1) * 5]
        ax.scatter(fan_x, fan_y, fan_z, c='red', marker='o', s=100)

    ax.set_xlabel('X axis')
    ax.set_ylabel('Y axis')
    ax.set_zlabel('Z axis')
    ax.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
    ax.set_title('Fan Positions in Greenhouse')

    plt.show()

# Output the optimal solution
best_individual = tools.selBest(population, k=1)[0]
print("Best Individual: ", best_individual)

# Plot the optimal fan configuration
```

```
plot_fan_positions(best_individual)
# Format output for the optimal configuration of each fan
num_fans = int(best_individual[0])
print("Number of Fans:", num_fans)
for i in range(num_fans):
    fan_config = best_individual[1 + i*5 : 1 + (i+1)*5]
    print(f"Fan {i+1} Configuration:")
    print(f" Speed: {fan_config[0]} m/s")
    print(f" Temperature: {fan_config[1]+20} C")
    print(f" Position: (X: {fan_config[2]}, Y: {fan_config[3]}, Z:
        {fan_config[4]})")


# Use the optimal solution to calculate temperature and wind speed
    distribution
best_individual = tools.selBest(population, k=1)[0]
num_fans = int(best_individual[0])
fan_parameters = best_individual[1:]
temperature_distribution = np.full((len(x), len(y), len(z)),
    ambient_temp)
wind_speed_distribution = np.zeros((len(x), len(y), len(z)))
for i in range(num_fans):
    fan_speed, fan_temp, fan_x, fan_y, fan_z = fan_parameters[i * 5:(i +
        1) * 5]
    temperature_distribution, wind_speed_distribution =
        update_temperature_and_wind_speed(
        temperature_distribution, wind_speed_distribution, fan_speed,
            fan_temp, fan_x, fan_y, fan_z, temp_decay_control,
            wind_decay_control)

# Plot 3D temperature distribution
fig = plt.figure(figsize=(18, 6))
ax1 = fig.add_subplot(131, projection='3d')
ax1.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
ax1.scatter(X.flatten(), Y.flatten(), Z.flatten(),
    c=temperature_distribution.flatten(), cmap='hot')
ax1.set_title('3D Temperature Distribution')
```

```
ax1.set_xlabel('Length (m)')
ax1.set_ylabel('Width (m)')
ax1.set_zlabel('Height (m)')

# Plot 3D wind speed distribution
ax2 = fig.add_subplot(132, projection='3d')
ax2.set_box_aspect([np.ptp(coord) for coord in [X, Y, Z]])
ax2.scatter(X.flatten(), Y.flatten(), Z.flatten(),
    c=wind_speed_distribution.flatten(), cmap='cool')
ax2.set_title('3D Wind Speed Distribution')
ax2.set_xlabel('Length (m)')
ax2.set_ylabel('Width (m)')
ax2.set_zlabel('Height (m)')

# Create 2D grids for temperature and wind speed distribution
x_2d, y_2d = np.meshgrid(np.linspace(0, length, 20), np.linspace(0,
    width, 40), indexing='xy')

# Plot temperature and wind speed distribution at a height of 0.5
    meters and at 0.1 meters
fig, axs = plt.subplots(2, 2, figsize=(12, 5))

# Temperature and Wind Speed Distribution at Height 0.5 meters
half_height_index = np.argmin(np.abs(z - 0.5))
temperature_half_height = temperature_distribution[:, :,
    half_height_index]
wind_speed_half_height = wind_speed_distribution[:, :,
    half_height_index]

c1 = axs[0, 0].contourf(x_2d, y_2d, temperature_half_height, cmap='hot')
plt.colorbar(c1, ax=axs[0, 0])
axs[0, 0].set_title('Temperature at 0.5 meters')
axs[0, 0].set_xlabel('Length (m)')
axs[0, 0].set_ylabel('Width (m)')

c2 = axs[0, 1].contourf(x_2d, y_2d, wind_speed_half_height, cmap='cool')
plt.colorbar(c2, ax=axs[0, 1])
```

```
axs[0, 1].set_title('Wind Speed at 0.5 meters')
axs[0, 1].set_xlabel('Length (m)')
axs[0, 1].set_ylabel('Width (m)')

# Temperature and Wind Speed Distribution at Height 0.5 meters
low_height_index = np.argmin(np.abs(z - 0.1))
temperature_low_height = temperature_distribution[:, :,
    low_height_index]
wind_speed_low_height = wind_speed_distribution[:, :, low_height_index]

c3 = axs[1, 0].contourf(x_2d, y_2d, temperature_low_height, cmap='hot')
plt.colorbar(c3, ax=axs[1, 0])
axs[1, 0].set_title('Temperature at 0.1 meters')
axs[1, 0].set_xlabel('Length (m)')
axs[1, 0].set_ylabel('Width (m)')

c4 = axs[1, 1].contourf(x_2d, y_2d, wind_speed_low_height, cmap='cool')
plt.colorbar(c4, ax=axs[1, 1])
axs[1, 1].set_title('Wind Speed at 0.1 meters')
axs[1, 1].set_xlabel('Length (m)')
axs[1, 1].set_ylabel('Width (m)')

plt.tight_layout()
plt.show()
```