VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

MATHEMATICAL MODELING (CO2011)

Assignment

"Dynamical systems in forecasting Greenhouse Micro-climate"

Advisor: Nguyễn An Khương
Nguyễn Tiến Thịnh
Students: Nguyễn Minh Hùng - 1952737(CC05,Team Leader)
Phạm Nhật Hoàng - 1952703 (CC05)
Nguyễn Bá Minh Hưng - 1952748 (CC05)
Nguyễn Tiến Dương - 1952639 (CC03)
Huỳnh Nhật Quang - 1952112 (CC03)

HO CHI MINH CITY, DECEMBER 2020

Contents

1 Member list & Workload

No.	Fullname	Student ID	Problems	Percentage of work
1	Nguyễn Minh Hùng	1952737	Ex2, Ex3, Ex4, Ex5 latex(ex2,3,4),edit and submit	- 40%
2	Phạm Nhật Hoàng	1952703	Ex2, Ex5(task ex2)	20%
3	Nguyễn Bá Minh Hưng	1952748	Ex5(task ex2,3,4) latex(ex 5)	- 20%
4	Huỳnh Nhật Quang	1952112	Ex1 latex(ex1)	- 10%
5	Nguyễn Tiến Dương	1952639	Ex1 latex(ex1)	- 10%

2 Background

2.1 Definition and classification

2.1.1 Definition

Mathematical models of scientific systems often lead to differential equations in which the independent variable is time. Such systems arise in astronomy, biology, chemistry, economics, engineering, physics and other disciplines. It is common to speak of a system of differential equations

$$\dot{x} = f(x)$$

as governing a dynamical system. A formal definition of dynamical system that is sometimes used in abstract studies is that of a family of maps $\phi(t, \cdot) : \Omega \to \Omega$, defined on a domain Ω parametrized by a variable t on an interval (a,b) and satisfying, for each p $\in \Omega$ the following three conditions:
(i) $\phi(0, p) = p$
(ii) $\phi \in C[(a, b) \times \Omega]$
(iii) $\phi(t + \alpha, p) = \phi(t, \phi(\alpha, p))$.

2.1.2 Classification according to different criteria:

Consider the three mentioned conditions:
If in item (ii) the C is replaced by C^k with $k \geqslant 1$, the dynamical system is said to be differentiable. These studies can be viewed as abstractions of flows of autonomous differential equations. When the mapping ϕ is generated by an initial-value problem

$$\dot{x} = f(x), x(t_0) = p$$

it is a homeomorphism, i.e., a continuous map with a continuous inverse. If the vector field f is C^k, it is a diffeomorphism (which is differentiable map that possesses a differential inverse.) There are also Discrete dynamical systems for which t takes on discrete values, but we shall consider continuous dynamical systems generated by autonomous system like equation $\dot{x} = f(x)$ - for which t takes on values in an interval of the real axis - unless otherwise specified.

2.2 General form of dynamical systems (first-order ODEs)

In the most general sense, a dynamical system is a tuple (T,M Φ) where T is a monoid, written additively, M is a non-empty set and Φ is a function
$\Phi : U \subseteq (T \times M) \to M$
with $proj_2$(U) = M (where $proj_2$ is the 2nd projection map)
I(x) = $\{t \in T : (t, x) \in U\}$
$\Phi(0, x) = x$
$\Phi(t_2, \Phi(t_1, x)) = \Phi(t_2 + t_1, x)$, for $t_1, t_2 + t_1 \in I(x)$ and $t_2 \in I(\Phi(t_1, x))$

The function Φ(t,x) is called the evolution funnction of the dynamical system: it associates to every point in the set M a unique image, depending on the variable t, called the evolution parameter. M is called phase space or state space. while the variable x represents an initial state of the system.

We often write
$\Phi_x(t) \equiv \Phi(t, x)$
$\Phi^t(x) \equiv \Phi(t, x)$
if we take one of the variables as constant.

$\Phi_x : I(x) \to M$
is called the flow through x and its graph trajectory through x.
The set
$\gamma_x \equiv \{\Phi(t, x) : t \in I(x)\}$
is called the orbit through x. Note that the orbit through x is the image of the flow through x.
A subset S of the state space M is called ϕ-**invariant** if for all x in S and all t in T
$\Phi(t, x) \in S$
Thus, in particular, if S is ϕ-**invariant**, I(x) = T for all x in S. That is, the flow through x must be defined for all time for every element of S.

In this assignment, we will discuss dynamical system of **First Order Differential Equations**. A first order system of n (not necessarily linear) equations in n unknown functions x1(t), x2(t), . . . , xn(t) in normal form is given by:

$$\begin{aligned} x_1'(t) &= f_1(t, x_1, x_2, ..., x_n) \\ x_2'(t) &= f_2(t, x_1, x_2, ..., x_n) \\ &... \\ x_n'(t) &= f_n(t, x_1, x_2, ..., x_n) \end{aligned}$$

And in this section, we will discuss on the linear system of ODEs which are continuous dynamical systems used in this assignment.

Consider the linear system in the normal form:

$$\begin{aligned} x_1'(t) &= a_{11}(t)x_1(t) + ... + a_{1n}(t)x_n(t) + f_1(t), \\ x_2'(t) &= a_{21}(t)x_1(t) + ... + a_{2n}(t)x_n(t) + f_2(t), \\ &... \\ x_n'(t) &= a_{n1}(t)x_1(t) + ... + a_{nn}(t)x_n(t) + f_n(t) \end{aligned}$$

In matrix and vector notations, we write it as

$$X'(t) = A(t)X(t) + F(t)$$

where $X(t) = [x_1(t)...x_n(t)]^T, F(t) = [f_1(t)...f_n(t)]^T$ and $A(t) = [a_{ij}(t)]$ is a n x n matrix.

The IVP (initial value problem) for the above system is to find a vector function $X(t) \in C^1$ that satisfies the system on an interval I and the initial conditions $X(t_o) = x_0 = (x_{1,0}, ..., x_{n,0})^T$, where $t_0 \in I$ and $x_0 \in R^n$.

2.3 Necessary and sufficient condition

According to the general form of system of linear ODEs we had discussed above, let's consider the following theorem:
Theorem: Given the initial value problem:

$$X' = F(X), X(t_0) = X_0$$

for $X_0 \in R^n$, suppose that $F : R^n -> R^n$ is C^1. Then, there exists a unique solution to this initial value problem. That is, there exists an a > 0 and a solution $X : (t_0 - a, t_0 + a) -> R^n$ of the differential equation such that $X(t_0) = X_0$.

We will not delve into the totality of the proof of this theorem in this report. Suffice to say, the proof relies on the technique of Picard iteration. The basic idea of this technique is to construct a sequence of functions which converges to the solution of the differential equation. The sequence of functions $p_k(t)$ is defined by $p_0(t) = x_0$, our initial condition, and

$$p_{k+1} = x_0 + \int_0^t p_k(s)\, ds \tag{1}$$

This technique is useful not only for proving this theorem, but also for approximating solutions to difficult or impossible to solve equations.

2.4 Examples

• Example 1: Suppose that a bread is removed from a 100^oC oven and placed in a room with a temperature of 25^oC. In 20 min the bread has a temperature of 60^oC.

We want to determine the time when the bread at a temperature of 89^oC.

Solution:
The rate of change of the temperature T(t) is the derivative $\frac{dT}{dt}$.
According to Newton's law of cooling, the rate at which the temperature T(t) changes in a cooling body is proportional to the difference between the temperature of the body and the constant temperature T_s of the surrounding medium. Symbolically, the statement is expressed as
$\frac{dT}{dt} = k(T - T_s)$
Where k is a constant of proportionality.
Separating the variables we have
$\frac{dT}{T - T_s} = kdt \iff \frac{dT}{T - 25} = kdt$

$\Rightarrow \int \frac{dT}{T - 25} = \int kdt \Rightarrow ln|T - 25| = kt + ln\ C$

$\Rightarrow T - 25 = e^{kt+lnC} = Ce^{kt}$

To find k we use the condition: if t = 20min then T = 60^oC. So

$60 - 25 = 75e^{k.20} \Rightarrow e^k = \left(\frac{7}{15}\right)^{\frac{1}{20}}$

$k = \frac{1}{20}.ln\left(\frac{7}{15}\right)$

Therefore

$T = 75\left(\frac{7}{15}\right)^{\frac{t}{20}} + 25$

We want to know when T(t) = 89^oC. Thus we solve

$89 = 75\left(\frac{7}{15}\right)^{\frac{t}{20}} + 25 \Rightarrow t \approx 4\ min$

• Example 2: Find the solution of the initial-value problem and calculate y(1).
y' + y = x, y(0) = 1.

Solution:
$P(x)= 1$,$Q(x)=x$.
We have the formula for this solution.

$\frac{dy}{dx} + P(x)y = Q(x)$

Mutiplying both sides of the aboved formula to $e^{\int P(x)dx}$, we have:

$e^{\int P(x)dx}.\frac{dy}{dx} + e^{\int P(x)dx}.P(x)y = e^{\int P(x)dx}.Q(x)$

Taking the primitive of both sides:

$e^{\int P(x)dx}.y = \int Q(x).e^{\int P(x)dx}dx + C$

$y = e^{-\int P(x)dx}.\left[\int Q(x).e^{\int P(x)dx}dx + C\right]$

Apply the formula mentioned above, we have

$y = e^{-\int 1dx}.\left[\int x.e^{\int 1dx}dx + C\right] = e^{-x}.\left[\int x.e^{x}dx + C\right]$

$= e^{-x}.[(x - 1).e^{x}dx + C] = (x - 1) + Ce^{-x}$

Since $x = 0,\ y = 1 \Rightarrow 1 = (0 - 1) + Ce^{0} \Rightarrow C = 2$. Therefore the solution to the initial-value problem is $y = x - 1 + 2e^{-x}$.
Finally, it's easy to find $y(1) = 1 - 1 + 2e^{-1} = 2e^{-1}$.

• Example 3: We will solve the example 2 with another method: Picard's Iteration Scheme.

Picard's Theorem: is proved by applying Picard's iteration scheme, which we now introduce. We begin by noticing that any solution to the initial value problem of Equations:

$\frac{dy}{dx} = f(x,y),\ y(x_o) = y_o$ (1)

It must also satisfy the integral equation:
$y(x) = y_o + \int_{x_o}^{x} f((t,y(t))dt$ (2)
because
$\int_{x_o}^{x} \frac{dy}{dt}dt = y(x) - y(x_o).$
The converse is also true: if y(x) satisfies Equation (2), then y' = ((t,y(t)) and $y(x_o) = y_o$. So Equation (1) may be replaced by Equation (2). This sets the stage for Picard's interation method: In the integrand in Equation (2), replace y(t) by the constant y_o, then integrate and call the resulting right-hand side of Equation (2) y_1(x):

$y_1(x) = y_o + \int_{x_o}^{x} f(t,y_o)dt$ (3)

This starts the process. To keep it goint, we use the iterative formulas

$y_{n+1}(x) = y_o + \int_{x_o}^{x} f(t,y_n(t))dt$ (4)

Solution:
Return to Example 2:
$y' = x - y,\ y(0) = 1$
For the problem at hand, $f(x,y) = x - y$. And Equation (3) becomes
$y_2(x) = 1 + \int_{0}^{x} f\left(t - 1 + t - x^2 + \frac{1}{6}t^3\right)dt = 1 - x + x^2 - \frac{1}{3}x^3 + \frac{1}{4!}x^4$
In this example it is possible to find the exact solution because
$\frac{dy}{dx} + y = x$

is a first-order differential equation that is linear in y. We have already shown how to find the general solution

$y = x - 1 + Ce^{-x}$

in the previous example. The solution of the initial value problem is then

$y = x - 1 + 2e^{-x}$

If we substitute the Maclaurin series for e^{-x} in this particular solution, we get

$$y = x - 1 + 2\left(1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \ldots\right) = 1 - x + x^2 - \frac{x^3}{3!} + 2\left(\frac{x^4}{4!} - \frac{x^5}{5!} + \ldots\right)$$

The following example illustrate the Picard iteration scheme, but in most practical cases the computations soon become too burdensome to continue. However, it is one way we could get an idea of how the solution behaves near the initial point.

Introduce and present the approximation steps of the Explicit Euler and Explicit Runge-Kutta of order 4 algorithms to solve general first-order differential equations.

2.4.1 Euler's method

Given a differential equation $\frac{dy}{dx} = f(x, y)$ and an initial condition $y(x_0) = y_0$, we can approximate the solution $y = y(x)$ by its linearization:

$$L(x) = y(x_0) + y^{'}(x_0)(x - x_0)$$

or

$$L(x) = y_0 + f(x_0, y_0)(x - x_0)$$

The function $L(x)$ gives a good approximation to the solution $y(x)$ in a short interval about x_0. The basis of Euler's method is to path together a string of linearization to approximate the curve over a longer stretch.

• **Step 1:** We know the point (x_0, y_0) lies on the solution curve. So from this point which lies exactly on the solution curve, we have obtained the point (x_1, y_1), which lies very close to the point $(x_1, y(x_1))$ on the solution curve by the equation:

$$y_1 = L(x_1) = y_0 + f(x_0, y_0)dx$$

• **Step 2:** Using the point (x_1, y_1) and the slope $f(x_1, y_1)$ on the solution curve through (x_1, y_1), we take the second step. Setting $x_2 = x_1 + dx$, we use the linearization of the solution curve through (x_1, y_1) to calculate:

$$y_2 = L(x_2) = y_1 + f(x_1, y_1)dx$$

• **Step 3:** Continuing in this fashion, we take the third step and so on. We build literally an approximation to one of the solution by following the direction of the slope field of the differential equation.

The step (dx) would be small enough to calculate a good approximation. Euler's method is easy to implement on a computer or calculator. A computer generates a table of numerical solutions to an initial value problem, allowing us to input x_0 and y_0, the number of step n, and the step size dx. It then calculates the approximate solution value $y_1, y_2 ... y_n$ in iterative fashion, as just described.

While Euler's method is simple to program on a computer, it is inherently unstable, only first order accurate, and requires very small Δt in order to achieve reasonable results.

Mathematicians and engineers have developed clever algorithms to modify the slope such that information is used from one or more values of t between t_n and t_{n+1}. These algorithms include the implicit Euler method and various kinds of predictor-corrector techniques, which can be formulated to first-, second-, or higher-order accuracy.

2.4.2 Runge-Kutta method

The Runge-Kutta technique is fourth-order accurate, and can be thought of as a kind of predictor-corrector technique in that the final value of y_{n+1} at $t = t_{n+1}$ is calculated as:

$$y_{n+1} = y_n + \Delta y_{final}$$

where increment Δy_{final} is a weighted average of four "trial increments," namely Δy_1, Δy_2, Δy_3, and Δy_4, evaluated from slopes calculated at $t = t_n$, $t_{n+\frac{1}{2}}$, $t_{n+\frac{1}{2}}$, and t_{n+1}, respectively. The final predicted value of y_{n+1} at $t = t_{n+1}$ is calculated as:

$$\Delta y_{final} = \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

- **Step 1:** Compute the slope at x_n:

$$k_1 = f(x_n, y_n)$$

- **Step 2:** Compute the slope at the midpoint from k_1:

$$k_2 = f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_1)$$

- **Step 3:** Compute the slope at the midpoint from k_2:

$$k_3 = f(x_n + \tfrac{1}{2}h, y_n + \tfrac{1}{2}hk_2)$$

- **Step 4:** Compute the slope at the endpoint from k_3:

$$k_4 = f(x_n + h, y_n + hk_3)$$

- **Step 5:** Compute y_{n+1}:

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

2.5 Using Explicit Euler and Explicit Runge-Kutta, give approximate values of the exact solutions of the above examples at time t_0, t_0+h, $t_0 + 2h$, ..., $t_0 + 5h$ with optional h.

2.5.1 Example 1:

Suppose that a bread is removed from a 100^oC oven and placed in a room with a temperature of 25^oC. In 20 min the bread has a temperature of 60^oC. We want to determine the temperature after 4 min
We have:

$$\frac{dT}{dt} = \frac{1}{20} ln \frac{7}{15}(T - 25)$$

• **Euler's method:**
The initial point is $t_0 = 0$ and $T_0 = 100$ with step $h = dt = 0.8$ (min)

$$T_1 = T_0 + \frac{1}{20}ln\frac{7}{15}(T - 25)dt = 100 + \frac{1}{20}ln\frac{7}{15}(100 - 25)0.8 = 97.714$$

$$T_2 = T_1 + \frac{1}{20}ln\frac{7}{15}(T - 25)dt = 97.714 + \frac{1}{20}ln\frac{7}{15}(97.714 - 25)0.8 = 95.497$$

$$T_3 = T_2 + \frac{1}{20}ln\frac{7}{15}(T - 25)dt = 95.497 + \frac{1}{20}ln\frac{7}{15}(95.497 - 25)0.8 = 93.348$$

$$T_4 = T_3 + \frac{1}{20}ln\frac{7}{15}(T - 25)dt = 93.348 + \frac{1}{20}ln\frac{7}{15}(93.348 - 25)0.8 = 91.264$$

$$T_5 = T_4 + \frac{1}{20}ln\frac{7}{15}(T - 25)dt = 91.264 + \frac{1}{20}ln\frac{7}{15}(91.264 - 25)0.8 = 89.244$$

As we can see the approximation is 89.244 compare with the exact number is 89 The error is 0.27%
• **Runge-Kutta's method:**

2.5.2 Example 2:

Find the solution of the initial-value problem and calculate $y(1)$.

$$y' + y = x,\ y(0) = 1$$

We have:

$$\frac{dy}{dx} = x - y$$

• **Euler's method:**
The initial point is $x_0 = 0$ and $y_0 = 1$ with step $h = dt = 0.2$

$$y_1 = y_0 + (x_0 - y_0)dx = 1 + (0 - 1) * 0.2 = 0.8$$

$$y_2 = y_1 + (x_1 - y_1)dx = 0.8 + (0.2 - 0.8) * 0.2 = 0.68$$

$$y_3 = y_2 + (x_2 - y_2)dx = 0.68 + (0.4 - 0.68) * 0.2 = 0.624$$

$$y_4 = y_3 + (x_3 - y_3)dx = 0.628 + (0.6 - 0.628) * 0.2 = 0.6192$$

$$y_5 = y_5 + (x_5 - y_5)dx = 0.6192 + (0.8 - 0.6192) * 0.2 = 0.65032$$

As we can see the approximation is 0.65032 compare with the exact number is $2e^{-1}$ The error is 11.61%
• **Runge-Kutta's method:**

Step 1

$$k_1 = f(0, 1) = 0 - 1 = -1$$

$$k_2 = f(0 + \frac{1}{2} * 0.2, 1 + \frac{1}{2} * 0.2 * (-1)) = -0.8$$

$$k_3 = f(0 + \frac{1}{2} * 0.2, 1 + \frac{1}{2} * 0.2 * (-0.8)) = -0.82$$

$$k_4 = f(0 + 0.2, 1 + 0.2 * (-0.82)) = -0.64$$

$$y(0.2) = 1 + \frac{0.2}{6}(-1 + 2 * (-0.8) + 2 * (-0.82) - 0.64) = 0.84$$

Step 2

$$k_1 = f(0.2, 0.84) = 0.2 - 0.84 = -0.64$$

$$k_2 = f(0.2 + \frac{1}{2} * 0.2, 0.84 + \frac{1}{2} * 0.2 * (-0.64)) = -0.48$$

$$k_3 = f(0.2 + \frac{1}{2} * 0.2, 0.84 + \frac{1}{2} * 0.2 * (-0.48)) = -0.49$$

$$k_4 = f(0.2 + 0.2, 0.84 + 0.2 * (-0.49)) = -0.34$$

$$y(0.4) = 0.84 + \frac{0.2}{6}(-0.64 + 2 * (-0.48) + 2 * (-0.49) - 0.34) = 0.74$$

Step 3

$$k_1 = f(0.4, 0.74) = 0.4 - 0.74 = -0.34$$

$$k_2 = f(0.4 + \frac{1}{2} * 0.2, 0.74 + \frac{1}{2} * 0.2 * (-0.34)) = -0.21$$

$$k_3 = f(0.4 + \frac{1}{2} * 0.2, 0.74 + \frac{1}{2} * 0.2 * (-0.21)) = -0.22$$

$$k_4 = f(0.4 + 0.2, 0.74 + 0.2 * (-0.22)) = -0.1$$

$$y(0.6) = 0.74 + \frac{0.2}{6}(-0.34 + 2 * (-0.21) + 2 * (-0.22) - 0.1) = 0.7$$

Step 4

$$k_1 = f(0.6, 0.7) = 0.6 - 0.7 = -0.1$$

$$k_2 = f(0.6 + \frac{1}{2} * 0.2, 0.7 + \frac{1}{2} * 0.2 * (-0.1)) = 0.01$$

$$k_3 = f(0.6 + \frac{1}{2} * 0.2, 0.7 + \frac{1}{2} * 0.2 * (-0.01)) = -0.001$$

$$k_4 = f(0.6 + 0.2, 0.7 + 0.2 * (-0.001)) = 0.1002$$

$$y(0.6) = 0.7 + \frac{0.2}{6}(-0.1 + 2 * 0.01 + 2 * (-0.001) + 0.1002) = 0.7$$

Step 5

$$k_1 = f(0.8, 0.7) = 0.8 - 0.7 = 0.1$$

$$k_2 = f(0.8 + \frac{1}{2} * 0.2, 0.7 + \frac{1}{2} * 0.2 * 0.097) = 0.19$$

$$k_3 = f(0.8 + \frac{1}{2} * 0.2, 0.7 + \frac{1}{2} * 0.2 * 0.19) = 0.18$$

$$k_4 = f(0.8 + 0.2, 0.7 + 0.2 * 0.18) = 0.26$$

$$y(0.4) = 0.703 + \frac{0.2}{6}(0.1 + 2 * 0.19 + 2 * 0.18 + 0.26) = 0.74$$

As we can see the approximation is 0.74 compare with the exact number is $2e^{-1}$ The error is 0.58%

3 Application

3.1 Exercise 2

3.1.1 Restate the model

3.1.1.a CO_2 exchange

In this section, the concentration of CO2 in the greenhouse air will be described in more detail. For general purposes, we consider a greenhouse equipped with thermal screens. Thermal screens are made of many different materials such as metal or elastic-plastic. They are used to protect crops from damage caused by direct sunlight as well as from freezing in winter.

A thermal screen divides a greenhouse into two different compartments, above and below the screen. The upper compartment is often narrower than the lower one (see Figure 3).This results

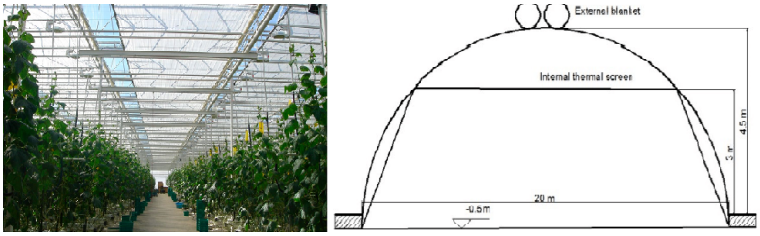


Figure 3: Thermal screens work to shielding trees and regulating the greenhouse climate.

different concentrations of CO2 in the greenhouse air above and below the screen. A schematic summary of the circulation of CO2 in the greenhouse is then shown in Figure 4.

For the lower compartment of the greenhouse, the amount of CO2 is mainly brought in from sources such as the natural airflow through the pad system and exited through the fan system (see Figure 5) and Figure 6). In addition, CO2 in this space is also received from direct air heaters (see Figure 7) and from the third party. A portion of CO2 in the lower compartment is also lost to the upper part of the greenhouse under the direction of the difference in temperature and air density between the two compartments while a large amount of CO2 in the lower compartment of the greenhouse is absorbed into plants for photosynthesis. For the upper part of the greenhouse, the amount of CO2 is mainly received from the exchange with the lower compartment and is released to the outside through the roof openings (if any) as in Figure 10.

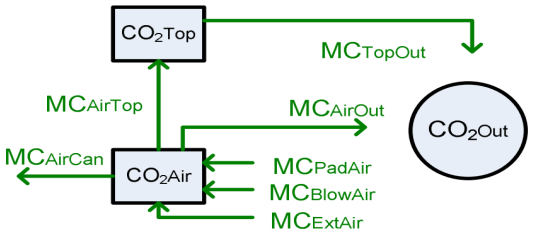


Figure 4: The CO_2 flow inside and outside a greenhouse.

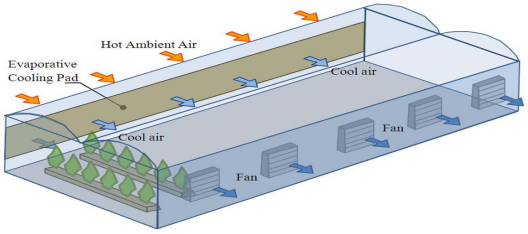


Figure 5: The movement of CO_2 through the pad system and fan system.

3.1.1.b Dynamical system models and assumptions

In this section, a dynamical system representing the CO2 concentration in the greenhouse will be discussed. The model has been studied by many authors [Mar94; Van11; De 96]. From Figure

Figure 6: Pad system (left) and fan system (right).

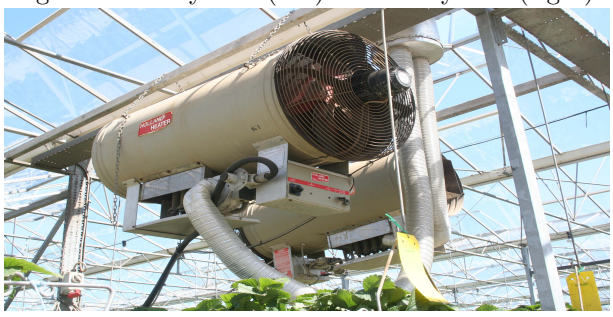


Figure 7: Heating system.

4, the two differential equations represents the fluctuation of CO_2 concentration in the lower and upper compartments of the greenhouse.

$$\begin{cases} cap_{CO_2\,Air}\dot{CO_2}_{Air} = MC_{BlowAir} + MC_{ExtAir} + MC_{PadAir} \\ \qquad\qquad\qquad\qquad -MC_{AirCan} - MC_{AirTop} - MC_{AirOut}, & (1) \\ cap_{CO_2\,Top}\dot{CO_2}_{Top} = MC_{AirTop} - MC_{TopOut}. & (2) \end{cases}$$

Assuming that the amount of CO2 in the air in the lower and upper space of the greenhouse is not affected by other elements except for those shown in Figure 4. Furthermore, the greenhouse is a perfect tank in the sense that the concentration of CO2 is uniformly distributed in each compartment of the greenhouse. The notations capA, CO2 A, CO_2 A and MC_{AB} are respectively the capacity to store CO_2 in A (m), the CO_2 concentration in A (mg m^3), the rate of change of CO_2 concentration in A (mg m^3 s^1), and the net CO_2 flux from A to B (mg m^2 s^1), where Air and Top represent the lower and upper compartments, Blow represents the direct air heater,

Ext represents the source from the third party, P ad represents the pad system, Can represents the total foliage of the plants inside the greenhouse, and Out represents the space outside the greenhouse.

Here are formulas to calculate MC_{AB}. First, we consider the amount of CO_2 going from the heater into the greenhouse air as follows.

$$MC_{BlowAir} = \frac{\eta_{HeatCO_2} U_{Blow} P_{Blow}}{A_{Flr}}. \tag{3}$$

In this formula, $\eta_{HeatCO2}$ is the amount of CO_2 generated when 1 Joule of sensible heat is generated by the heater (mg $CO2\ J^1$). The dimensionless parameter U_{Blow}, which is to control the amount of CO_2 generated by the heater, is a number in [0, 1]. The factor P_{Blow} is the CO_2-generation capacity of the heater (W) and A_{Flr} is the area of the greenhouse (m^2).

Similarly, the amount of CO_2 that is pumped into the greenhouse by the third party that supplies CO_2 is given by

$$MC_{ExtAir} = \frac{U_{ExtCO_2} \phi_{ExtCO_2}}{A_{Flr}}. \tag{4}$$

The notations U_{ExtCO2} and ϕ_{ExtCO2} are respectively a dimensionless parameter in [0, 1] that adjusts the rate at which the gas is injected into the greenhouse and that represents the third party's ability to pump CO_2 (mg s^1).

On the other hand, the amount of CO_2 that enters the greenhouse through the pad system is due to the difference in the concentration of CO_2 inside and outside the greenhouse and the ability of the pad system for the air to go through as in Figure 6. Furthermore, the pad can be adjusted to let in more air. The following formula is used to calculate MC_{PadAir}.

$$MC_{PadAir} = f_{Pad}(CO_{2\,Out} - CO_{2\,Air}) = \frac{U_{Pad}\phi_{Pad}}{A_{Flr}}(CO_{2\,Out} - CO_{2\,Air}). \tag{5}$$

The flux f_{Pad} (m s^1) is the product of the dimensionless parameter UP ad in [0, 1], which represents the permeability of the pad and ϕ_{Pad}, which is the ability for the airflow to pass through ($m^3\ s^1$) divided by the area of the greenhouse floor.

The net flux of CO_2 from the lower compartment to the upper compartment of the greenhouse is more complicated and it depends on the difference in temperature and air density between the two compartments.

$$MC_{AirTop} = f_{ThScr}(CO_{2\,Air} - CO_{2\,Top}), \tag{6}$$

where the airflow rate through the thermal screen f_{ThScr} (m s^1) is the sum of the penetration rate through the screen and the rate at the open regions that are not covered by the thermal screen (see Figure 8).

$$f_{ThScr} = U_{ThScr} K_{ThScr} |T_{Air} - T_{Top}|^{\frac{2}{3}} + (1 - U_{ThScr}) \left[\frac{g(1 - U_{ThScr})}{2\rho_{Air}^{Mean}} |\rho_{Air} - \rho_{Top}| \right]^{\frac{1}{2}}. \tag{7}$$

The dimensionless parameter $U_{ThScr} \in [0, 1]$ represents the places covered by the thermal

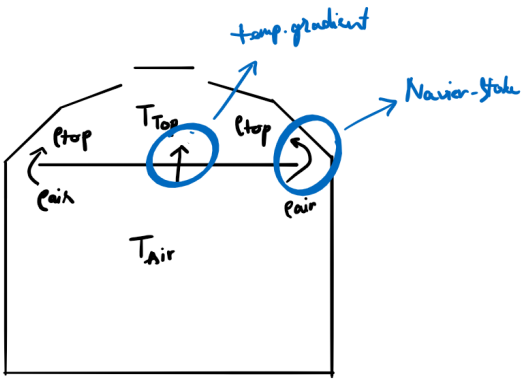


Figure 8: Movement of air through the thermal screen.

screen. The flux through those places depends on the difference between the temperature above the screen T_{Top} (K) and the temperature below the screen T_{Air} (K) and the permeability of the screen K_{ThScr} (m $K^{\frac{-2}{3}}$ s^1). At the places that are not covered by the thermal screen, the flux is given by a Navier–Stokes equation depending on the difference of the air density below the screen ρ_{Air} and the air density above the screen ρ_{Top} (kg m^3). Note that the coefficient 2/3 in the formula (7) comes from the experiment in the work of [Bal89]. In that work, the authors used measured data on the air-exchange rate through different kinds of thermal screens, which are made of 12 different materials, to train the model $K_{ThScr}|T_{Air}T_{Top}|^m$ where m is an adjustable parameter to find that m is nearly $0.66 \approx 2/3$. Particularly, the Navier–Stokes formula comes from the study of [MG], in which the authors considered the theoretical model of air exchange through cracks in the screen surface caused by the air-density difference

$$\phi_{crack} = \frac{L \cdot SO}{\rho_{mean}} \left[\frac{1}{2} \rho_{mean} \cdot SO \cdot g \cdot (\rho_1 - \rho_2) \right]^{\frac{1}{2}}, \tag{8}$$

where ϕ_{crack} (m s^1) is the rate of airflow through the screen, L (m) is the length of the opening on the screen, SO is the percentage of the opening on the screen (m), ρ_{mean} (kg m^3) is the

average density of the air density upper the screen ρ_1 (kg m^3) and the air density beneath the screen ρ_2 (kg m^3), and g is the gravitational acceleration (m s^2). The following and experimental works have also shown that the Navier–Stokes formula (8) gives good results once compared with the measured data.

Similarly, for the net CO_2 flux from the inside to the outside of the greenhouse, let consider the following formula

$$MC_{AirOut} = (f_{VentSide} + f_{VentForced})(CO_{2\,Air} - CO_{2\,Out}). \tag{9}$$

Here, $f_{VentSide}$ is the flux due to the fan system on the sidewalls of the greenhouse and $f_{VentForced}$ is the flux due to the fan system inside the greenhouse. Both of them are measured in m s^1. In this case, the Bernoulli principle plays an important role represented by the pressure difference from the outside of the greenhouse, which is caused by the natural airflow through the roof surface, and the pressure from the inside of the greenhouse caused by the lateral airflow as in (see Figure 9). The Stack effect, also known as the Chimney effect, should also be considered (see Figure 10). The Stack effect is an effect where in winter, cold air flows from outside into the greenhouse and is gradually heated by the heating system and tends to go up and escape back to the outside through the roof openings, in the summer it is the opposite.

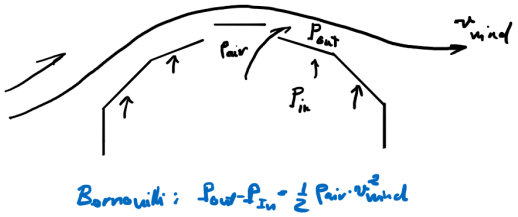


Figure 9: Airflow through the greenhouse openings.

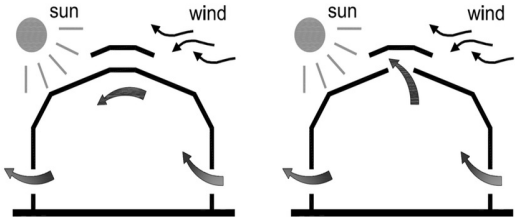


Figure 10: No Stack effect (left) and Stack effect (right).

To generalize the model for many different types of greenhouses, the following general formula $f_{VentRoofSide}$ (m s^1) is used to set the formula for $f_{VentSide}$ [Kit+96].

$$f_{VentRoofSide} = \frac{C_d}{A_{Flr}} \left[\frac{U^2_{Roof} U^2_{Side} A^2_{Roof} A^2_{Side}}{U^2_{Roof} A^2_{Roof} + U^2_{Side} A^2_{Side}} \cdot \frac{2gh_{SideRoof}(T_{Air} - T_{Out})}{T^{Mean}_{Air}} + \left(\frac{U_{Roof} A_{Roof} + U_{Side} A_{Side}}{2} \right)^2 C_w v^2_{Wind} \right]^{\frac{1}{2}}. \tag{10}$$

The formula (10) is the sum of the two components multiplied by the ratio between the dimensionless discharged coefficient C_d and the area of the greenhouse floor A_{Flr} (m^2). The first

component, depending on the temperature difference between the outside and the inside of the greenhouse (in the space under the thermal screen), represents the Stack effect when the ventilation area on the roof A_{Roof} (m^2) is non-zero. The second component is given by the pressure difference inside and outside the greenhouse and is calculated as the total area of the ventilation openings on the greenhouse divided by two times the natural wind speed v_{Wind} (m s^1) squared and the global wind pressure coefficient C_w (dimensionless). The coefficients C_d and C_w are theoretical coefficients depending on the structure and shape of the greenhouse and can be estimated by using experimental data.

In addition, this topic also explores insect screens on ventilation openings and ventilators and the leakage coefficient of the greenhouse. In the presence of an insect screen, the movement speed of the air currents through the ventilation areas will be reduced by a factor

$$\eta_{InsScr} = \zeta_{InsScr}(2 - \zeta_{InsScr}), \tag{11}$$

where ζ_{InsScr} is the porosity (dimensionless), which is the ratio of the area of the holes in the screen to the total area of the screen. Given the leakage coefficient $c_{leakage}$, which depends on the greenhouse type and is dimensionless, the air-exchange rate is added an amount of approximately 50% of the leakage rate

$$f_{leakage} = \begin{cases} 0.25 \cdot c_{leakage}, & v_{Wind} < 0.25, \\ v_{Wind} \cdot c_{leakage}, & v_{Wind} \geq 0.25. \end{cases} \tag{12}$$

Implicitly, we assumed the uniform distribution of the leakage rate.

Let $\eta_{SideThr}$ be the Stack-effect threshold. If η_{Side}, the ratio between the sidewalls ventilation area and the total ventilation area, exceeds the threshold, the Stack effect does not occur and vice versa. Then, $f_{VentSide}$ is given by the following

$$f_{VentSide} = \begin{cases} \eta_{InsScr} f''_{VentSide} + 0.5 f_{leakage}, & \eta_{Side} \geq \eta_{Side\_Thr}, \\ \eta_{InsScr} \Big[U_{ThScr} f''_{VentSide} & \\ \quad + (1 - U_{ThScr}) f_{VentRoofSide} \eta_{Side} \Big] + 0.5 f_{leakage}, & \eta_{Side} < \eta_{Side\_Thr}. \end{cases} \tag{13}$$

In which, $f''_{VentSide} = \frac{C_d U_{Side} A_{Side} v_{Wind}}{2 A_{Flr}} \sqrt{C_w}$ when $A_{Roof} = 0$. Moreover, the Stack effect does not occur where is covered by the thermal screen. In case when you can't get the value of $\eta_{SideThr}$, you can use the $\eta_{RoofThr}$ instead.

The flux $f_{VentForced}$ by the fan system inside the greenhouse is calculated as follows.

$$f_{VentForced} = \frac{\eta_{InsScr} U_{VentForced} \phi_{VentForced}}{A_{Flr}}. \tag{14}$$

The dimensionless parameter $U_{VentForced} \in [0, 1]$ is to adjust the wind speed $\phi_{VentForced}$ due to the system ($m^3\ s^1$).

Similarly to MC_{AirOut}, the net CO_2 flux from the greenhouse to outside the greenhouse through the roof openings is calculated by using the formula

$$MC_{TopOut} = f_{VentRoof}(CO_{2\,Top} - CO_{2\,Out}). \tag{15}$$

where $f_{VentRoof}$ is the flux rate through the roof openings and is given by

$$f_{VentRoof} = \begin{cases} \eta_{InsScr} f''_{VentRoof} + 0.5 f_{leakage}, & \eta_{Roof} \geq \eta_{Roof\_Thr}, \\ \eta_{InsScr}\Big[U_{ThScr} f''_{VentRoof} & \\ \quad +(1 - U_{ThScr}) f_{VentRoofSide} \eta_{Side}\Big] + 0.5 f_{leakage}, & \eta_{Roof} < \eta_{Roof\_Thr}. \end{cases} \tag{16}$$

However, this $f_{VentRoof}$ differs from $f_{VentSide}$ as in (13), when the ratio η_{Roof} of the roof opening area to the total ventilation area exceeds the Stack-effect threshold $\eta_{RoofThr}$, the Stack effect does not occur and we cannot reuse the formula $f_{VentRoofSide}$ in (10) where $A_{Side} = 0$ to calculate $f''_{VentRoof}$, which is introduced in [BB95].

$$f''_{VentRoof} = \frac{C_d U_{Roof} A_{Roof}}{2A_{Flr}} \left[\frac{g h_{Roof}(T_{Air} - T_{Out})}{2T^{Mean}_{Air}} + C_w v^2_{Wind}\right]^{\frac{1}{2}}. \tag{17}$$

Finally, we need to describe the amount of CO_2 that is absorbed into the leaves due to photosynthesis.

$$MC_{AirCan} = M_{CH_2O} h_{C_{Buf}} (P - R). \tag{18}$$

Here, M_{CH2O} is the molar mass of CH_2O (mg μmol^1), P is the photosynthetic rate (μmol {CO2} $m^2\ s^1$), R is the respiration rate (μmol {CO2} $m^2\ s^1$), and

$$h_{C_{Buf}} = \begin{cases} 0, & C_{Buf} > C^{Max}_{Buf}, \\ 1, & C_{Buf} \leq C^{Max}_{Buf}, \end{cases} \tag{19}$$

shows the cessation of photosynthesis when CH_2O is C_{Buf} (mg {CO2} m^2) has reached C^{Max}_{Buf} (mg $\{CH_2O\}\ m^{-2}$), which is the limit of the carbohydrates storage of the plants. Usually, the respiration rate is negligible compared to the photosynthetic rate and can be omitted or calculated as about 1% of the photosynthetic rate. The photosynthetic rate will be described in more detail in the next section.

To simplify the assignment, $h_{C_{Buf}}$ will always have a value of 1, meaning that C_{Buf} will have no effect on the CO_2 fluctuation.

3.1.1.c Photosynthesis of C3 plants

In this topic, we only consider plants belonging to the C3 group including tomatoes, cucumbers, etc. Photosynthesis is the process of using CO_2, water, and energy from sunlight to form organic compounds that feed plants. This process is mainly done by chlorophyll contained in chloroplasts, a special kind of organelles of leaf and plant cells. Photosynthesis has two phases consisting of the light-dependence phase and the light-independence (or dark) phase. In the lightdependence phase, the leaves absorb sunlight and convert to energy in the thylakoid component of the chloroplasts to provide energy for the dark phase. Products of the light-dependence phase are NADPH (Nicotinamide Adenine Dinucleotide Phosphate) and ATP (Adenosine Triphosphate). In the dark phase, also known as the Calvin cycle, a sequence of biochemical reactions as CO_2 assimilation, CO_2 reduction, and the regeneration of CO_2 receptor, which is the so-called Rubisco enzyme contained in the stroma of the chloroplasts, without the need of light.

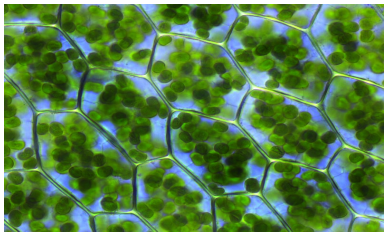


Figure 11: Chloroplasts.

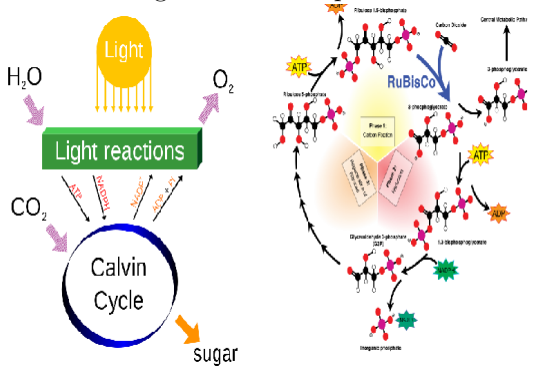


Figure 12: Light-dependence and light-independence phases of photosynthesis.

a/ Photosynthesis model for one leaf unit

There are many ways to model the rate of the photosynthesis of the plants. In this framework, we will combine more than one model if necessary (see [Lom+75] for more details).

b/The diffusion of CO_2 into the leaves

The photosynthetic rate P per leaf unit can be seen as the rate at which CO2 diffuses from the air into the leaf cells through stomata that are scattered on both sides of leaves as shown in Fig13.

The diffusion process is represented by Fick's law given by

$$P = \frac{CO_{2\,Air} - CO_{2\,Stom}}{Res}. \tag{20}$$

The notation CO_{2Stom} is the concentration of CO_2 in the stomata ($\mu mol\ m^3$) and Res is the resistance-to-absorption coefficient (s m^1). This coefficient of resistance depends on many factors including the speed of the wind blowing through the leaves.

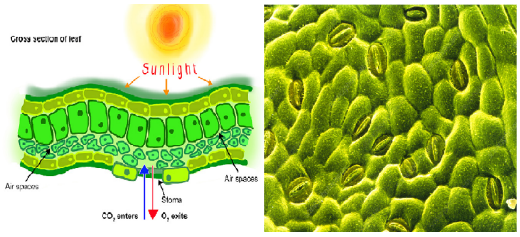


Figure 13: Diffusion of CO_2 into the leaf cells (left) and stomata (right).

c/ Biochemistry in the dark phase

Michaelis–Menten kinetic models, named after the German biologist Leonor Michaelis and the Canadian physicist Maud Menten, can be used to represent the biochemical process in the dark phase of the photosynthesis. The process occurs in the chloroplasts when the amount of CO_2 absorbed into the stomata and the Rubisco enzyme present in the chloroplasts form an unstable complex. This, in turn, quickly degenerates into Rubisco and another substance. For example, in the first step, water will be separated into 4 cations H^+ and 4 free electrons with the accompanying product of O_2. In the second step, CO_2 in the substrate is combined with the free electrons and the H^+ cations to produce carbohydrates and water again.

$$2H_2O \rightarrow 4e^- + 4H^+ + O_2,$$
$$CO_2 + 4e^- + 4H^+ \rightarrow CH_2O + H_2O.$$

Let K_M be the substrate concentration in the case where the reaction rate is exactly 50% of the reaction rate at the saturation point. Then, for given substrate concentration, the reaction rate can be obtained by multiplying the rate at the saturation point (or the maximum rate) by the reciprocal of the sum of 1 and the ratio of K_M to the considered substrate concentration (see Figure 14).

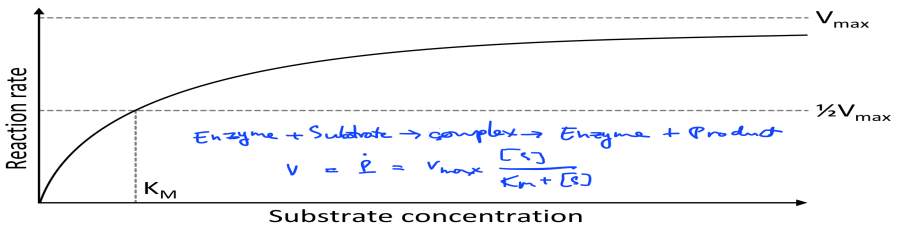


Figure 14: Michaelis–Menten kinetic model.

Then, the rate of the photosynthesis of the plants is given by

$$P = \frac{P_{Max} \cdot CO_{2\,Stom}}{CO_{2\,Stom} + CO_{2\,0.5}}. \tag{21}$$

where $CO_{20.5}$ is the concentration of CO_2 in the substrate when P = $P_Max/2$ ($\mu mol\ m^3$). Solve for CO_{2Stom}, from (20) and (21), the photosynthetic rate P satisfies the equation

$$ResP^2(CO_{2\,Air} + CO_{20.5} + ResP_{Max})P + CO_{2\,Air}P_{Max} = 0. \tag{22}$$

For the quadratic equation above, the focus is in the solution P such that $P \to P_{Max}$ as $CO_{2Air} \to +\infty$. Noting that the photosynthetic rate P no longer depends on the concentration of CO_2 in the stomata but only on the concentration of CO_2 in the air, the resistance coefficient Res, and the maximum photosynthetic rate.

d/ Maximum rate of photosynthesis

To solve the equation (22), the maximum rate of photosynthesis needs to be determined. For the model for the photosynthesis of one leaf unit, the maximum rate of photosynthesis is taken as a function of the leaf temperature, activation energy, and deactivation energy. Usually, that rate will be determined by the chemical reaction Arrhenius model

$$k(T) = k(T_0)e^{-\frac{H_a}{R}\left(\frac{1}{T} - \frac{1}{T_0}\right)}, \tag{23}$$

where $k(T)$ is the reaction rate at T (K), T_0 is the optimum temperature for which the reaction rate is known (K), Ha is the activation energy for the reaction (J mol^1), and R is the ideal gas constant (J mol^1 K^1).

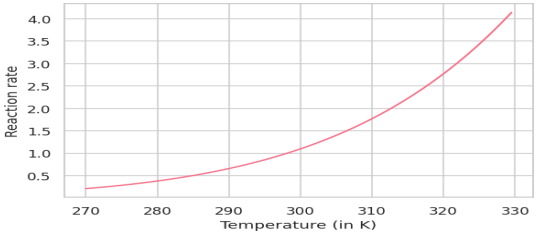


Figure 15: Arrhenius model with $T_0 = 298.15$, $k(T_0) = 1$, and $H_a = 37000$.

However, when the temperature is increasing, up to a certain threshold, the enzyme activity will be inhibited and the photosynthesis is slowed down and stop. Thus, the Arrhenius model is not sufficient to explain the inhibition of the enzyme and the following model is seen as the model for the activity of the Rubisco enzyme during photosynthesis and depends on the leaf temperature.

$$f(T) = \frac{1+e^{-\frac{H_d}{R}\left(\frac{1}{T_0} - \frac{1}{\frac{H_d}{S}}\right)}}{1+e^{-\frac{H_d}{R}\left(\frac{1}{T} - \frac{1}{\frac{H_d}{S}}\right)}} \tag{24}$$

In the model (24), $f(T)$ represents the enzyme activity at T (K), H_d is the deactivation energy (J mol^1), and S is the corresponding entropy quantity (J mol^1 K^1). By combining the model (23) and (24), the maximum rate of photosynthesis per leaf unit is given by the formula

$$P_{Max}(T) = k(T)f(T). \tag{25}$$

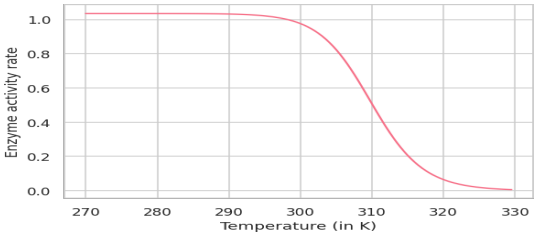


Figure 16: Enzyme activity model with $H_d = 220000$ and $S = 710$.

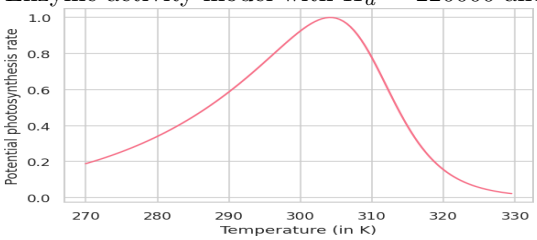


Figure 17: Maximum photosynthetic rate $P_Max(T) = k(T)f(T)$ (normalized).

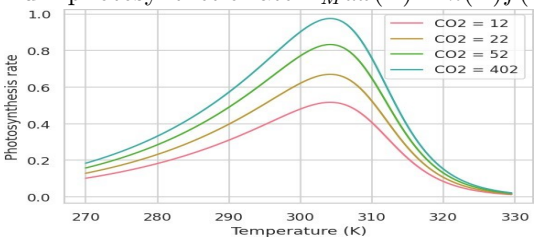


Figure 18: Photosynthetic rate for different values of CO_2 concentration in the greenhouse air and resistance coefficient $Res = 2.5$ (normalized).

e/ Photosynthesis model for the whole canopy

In this section, we will develop a model for the whole foliage of the plants inside the greenhouse.

f/ Leaf area index

First, we need to consider the concept of the leaf area index (LAI). The index LAI is calculated by the total leaf density per unit area of soil in a greenhouse. Then, the thicker the canopy is, the higher the index LAI is (see Figure 19). This index is very important for the canopy photosynthesis model because light absorbance is closely dependent on LAI. Due to Beer's law,

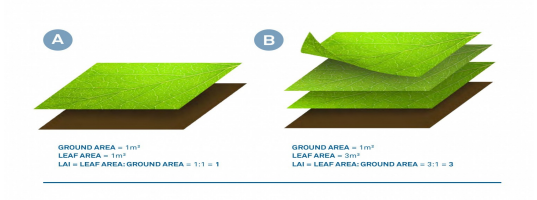


Figure 19: Leaf area index

if the light intensity before entering the canopy is I0 (μmol {photons} m^2 s^1), the intensity of the transmitted beam I (μmol {photons} m^2 s^1) is equal to

$$I = \frac{I_0.K.e^{-K.LAI}}{1-m} \qquad (26)$$

If the leaves are horizontally stratified such as in the case of tomato, the dimensionless extinction coefficient K will be between 0.7 and 1.0. Meanwhile, if the leaves are sloping as in the case of wet rice, K will be between 0.3 and 0.5. m is the transmittance coefficient of the leaves which is set as 0.1. Hence, the amount of light absorbed by the canopy can be measured as the difference in the intensity of the light ray before entering the foliage and after passing through the foliage

$$L = L_0(1 - \frac{K.e^{-K.LAI}}{1-m}) \tag{27}$$

In this formula, L is luminous flux received by the leaves per unit area of the greenhouse floor (μmol {photons} m^2 s^1) and L_0 is the initial value of L before going through the canopy. Noting that the formula (27) does not take into account the light reflection factor and the absorption of radiation from greenhouse objects. A more complete formula can be found in [Van11].

g/ The modified Arrhenius formula

To calculate the value of P_{Max}, which is the maximum photosynthetic rate of all leaves in the greenhouse, we consider the modified Arrhenius model (25) instead of (23).

$$k(T) = LAI.k(T_0).e^{-\frac{Ha}{R}(\frac{1}{T}-\frac{1}{T_0})} \tag{28}$$

Here, $k(T)$ is the reaction rate for the whole canopy at T (K) and $k(T_0)$ is the reaction rate under the optimal condition T_0 (K) of one leaf unit, and Ha is also the activation energy for one leaf unit.

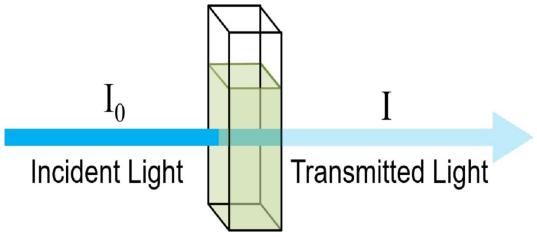


Figure 20: The incident-ray intensity decreases after passing through a liquid.

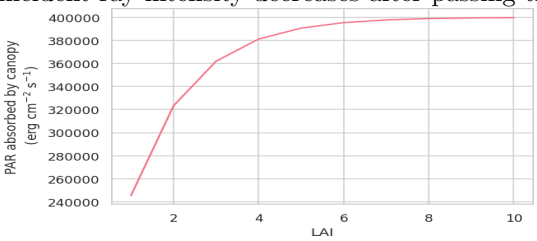


Figure 21: Photosynthetically active radiation.

h/ Michaelis–Menten kinetic model for P_{Max}

Unlike the photosynthesis model for one leaf unit, the amount of light energy absorbed into the foliage in response to LAI needs to be added since it affects the maximum photosynthetic rate P_{Max}. Therefore, we consider the following formula of P_{Max}, which is a dependent function on L and T.

$$P_{Max}(L,T) = \frac{P_{MLT}.P_{Max}(T).L}{L+L_{0.5}} \tag{29}$$

In which, $L_{0.5}$ is light intensity when $P_{Max}(L,T) = P_{Max}(T)/2(\mu mol$ {photons} m^{-2} $s^{-1})$, $P_{Max}(T)$ is calculated by using the formula (25) with $k(T)$ as in (28), and P_{MLT} are the maximumphotosynthetic rate at the point of light saturation and the optimal temperature T. Usually P_{MLT} is determined based on experiments and empirical works.

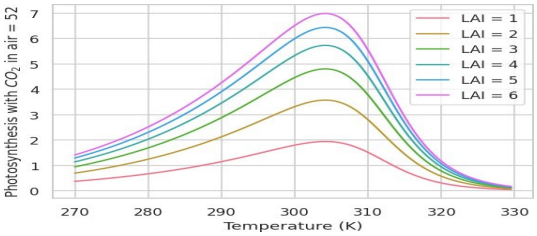


Figure 22: Photosynthesis of the canopy at a fixed value of CO_2 concentration and different LAI.

3.1.2 Calculation

According to the data we received from [Van11], we had listed some coefficients and const-variables in the table 1:

coeff. and var.	value	unit
ηHeatCO2	0.057	mgCO2/J
UBlow	0.5	none
PBlow	0.5 x 10^6	W
AFlr	1.4 x 10000	m^2
UExtCO2	0.5	none
ϕExtCO2	7.2 x 10^4	none
UPad	0.5	none
ϕPad	0	(m^3/s)
KThScr	0.05 x 10^{-3}	$m^3 m^{-2} K^{-0.66} s^{-1}$
g	9.81	m/s^2
ρAir	1.215	kg/m^3
ρTop	1.2	kg/m^3
CAPco2Air	4	m
CAPco2Top	0.8	m
Cd	0.75	none
Cw	0.09	none
ARoof	0.14 x 10000	m^2
ASide	0	m^2
USide	0.5	none
hSideRoof	0	
Uventforced	0.5	none
ϕventforced	0	none
ηSide	1.1	none
ηSideThr	0	none
ζInsScr	1	none
Cleakage	10^{-4}	
ηRoof	1.3	none
ηRoofThr	0.9	none
M_CH2O	30 x 10^{-3}	mg/mol
hRoof	1.6	
TCanK	296	K
T25K	298.15	K
LAI	2.0	leafs/m^2
Jmax25	210	leafs.J
ηCO2airStom	0.67	mol

Bảng 1: Coefficients and variables (The Netherlands).

We had build a program to calculate the net CO2 flux from one place to an other place, the following Python code is build to present the functions:

```
1 def MCBlowAir(nHeatCO2, UBlow, PBlow, AFlr):
```

```
    return nHeatCO2 * UBlow * PBlow / AFlr

# formula 4
def MCExtAir(UExtCO2, phiExtCO2, AFlr):
    return UExtCO2 * phiExtCO2 / AFlr

# formula 5
def MCPadAir_1(fPad, CO2Out, CO2Air):
    return fPad * (CO2Out - CO2Air)

def MCPadAir_2(UPad, phiPad, AFlr, CO2Out, CO2Air):
    fPad = UPad * phiPad / AFlr
    return fPad * (CO2Out - CO2Air)

# formula 6
def MCAirTop(f_ThScr, CO2Air, CO2Top):
    return f_ThScr * (CO2Air - CO2Top)

# formula 7
def fThScr(UThScr, KThScr, TAir, TTop, g, pAir, pTop):
    a = UThScr * KThScr * pow(abs(TAir - TTop), 2 / 3)
    PMean_Air = (pAir + pTop) / 2
    b = (1 - UThScr) * pow(g * (1 - UThScr) * abs(pAir - pTop) / (2 * PMean_Air), 1 / 2)
    return a + b

# formula 9
def MCAirOut(f_VentSide, f_VentForce, CO2Air, CO2Out):
    return (f_VentSide + f_VentForce) * (CO2Air - CO2Out)

# formula 10
def fVentRoofSide(Cd, AFlr, URoof, USide, ARoof, ASide, g, hSideRoof, TAir, TOut, Cw, vWind):
    a = Cd / AFlr
    b = pow(URoof * USide * ARoof * ASide, 2) / (pow(URoof * ARoof, 2) + pow(USide * ASide, 2) + epxilon)
    TMean_Air = (TAir + TOut) / 2
    c = 2 * g * hSideRoof * (TAir - TOut) / TMean_Air
    _d = (URoof * ARoof + USide * ASide) / 2
    d = pow(_d, 2) * Cw * pow(vWind, 2)
    return a * sqrt(b * c + d)

# formula 11
def nInsScr(sInsScr):
    return sInsScr * (2 - sInsScr)

# formula 12
def fleakage(cleakage, vWind):
    if vWind < 0.25:
        return 0.25 * cleakage
    else:
        return vWind * cleakage

# ppfVentSide with no stack eff
def ppfVentSide(Cd, USide, ASide, vWind, AFlr, Cw):
    return Cd * USide * ASide * vWind * sqrt(Cw) / (2 * AFlr)


# formula 13
def fVentSide(n_InsScr, ppf_VentSide, f_leakage, UThScr, ppf_VentRoofSide, nSide, nSide_Thr):
    if nSide >= nSide_Thr:
```

```
60          return n_InsScr * ppf_VentSide + 0.5 * f_leakage
61      else:
62          return n_InsScr * (UThScr * ppf_VentSide + (1 - UThScr) * ppf_VentRoofSide
        * nSide) + 0.5 * f_leakage
63
64
65  # formula 14
66  def fVentForced(n_InsScr, UVentForced, phiVentForced, AFlr):
67      return n_InsScr * UVentForced * phiVentForced / AFlr
68
69
70  # formula 15
71  def MCTopOut(f_VentRoof, CO2Top, CO2Out):
72      return f_VentRoof * (CO2Top - CO2Out)
73
74
75  # formula 16
76  def fVentRoof(n_InsScr, f_leakage, UThScr, ppf_VentRoofSide, nRoof, nSide,
        nRoof_Thr, ppf_VentRoof):
77      if nRoof >= nRoof_Thr:
78          return n_InsScr * ppf_VentRoof + 0.5 * f_leakage
79      else:
80          return n_InsScr * (UThScr * ppf_VentRoof + (1 - UThScr) * ppf_VentRoofSide
        * nSide) + 0.5 * f_leakage
81
82
83  # formula 17
84  def ppfVentRoof(Cd, URoof, ARoof, AFlr, g, hVent, TAir, TOut, Cw, vWind):
85      TMeanAir = (TAir + TOut) / 2
86      a = Cd * URoof * ARoof / (2 * AFlr)
87      b = g * hVent * (TAir - TOut) / 2 / TMeanAir + Cw * pow(vWind, 2)
88      return a * sqrt(b)
89
90
91  # formula 18, 19
92  def P(CO2Air, LAI):
93      T_Can_K = 20 + 273
94      J_POT = LAI * 210 * math.exp(37000 * (T_Can_K - 298.15) / (8.314 * T_Can_K *
        298.15)) * (
95                  1 + math.exp((710 * 298.15 - 220000) / (8.314 * 298.15))) / (
96                          1 + math.exp((710 * T_Can_K - 220000) / (8.314 * T_Can_K))
        )
97      J = (J_POT + 38.5 - math.sqrt(math.pow(J_POT + 38.5, 2) - 2.8 * J_POT * 38.5))
         / 1.4
98      CO2Stom = 0.67 * CO2Air
99      return (J * (CO2Stom - 498.1)) / (4 * (CO2Stom + 2 * 498.1))
100
101
102 def R(CO2Air, P):
103     CO2Stom = 0.67 * CO2Air
104     return P * 498.1 / CO2Stom
105
106
107 def MCAirCan(P, R, CBuf, CMaxBuf):
108     MCH2O = 0.03
109     hCBuf = 1
110     if CBuf > CMaxBuf:
111         hCBuf = 0
112     return MCH2O * hCBuf * (P - R)
```

Listing 1: Net flux Functions

Using these functions and some available coefficients, variables in table 1, we had build a function named **dx** with 2 parameters(CO2 Air and CO2 Top). These parameters are respectively CO2 concentration in the lower and upper compartments of the green house. The function **dx** is built based on 2 sub-functions **dxCO2Air, dxCO2Top** and they present for a dynamical system with 2 ODEs:

The following Python code is written to present for f(CO2_Air, CO2_Top)

```
# formula 1
def dxCO2Air(CO2Air, CO2Top):
    # Calculate MCBlowAir
    nHeatCO2 = N_HEAT_CO2
    UBlow = U_BLOW
    PBlow = P_BLOW
    AFlr = A_FLR
    MC_BlowAir = MCBlowAir(nHeatCO2, UBlow, PBlow, AFlr)

    # Calculate MCExtAir
    UExtCO2 = U_EXT_CO2
    phiExtCO2 = O_EXT_CO2
    MC_ExtAir = MCExtAir(UExtCO2, phiExtCO2, AFlr)

    # Calculate MCPadAir
    UPad = U_PAD
    phiPad = O_PAD
    CO2Out = CO2_OUT
    MCPadAir = MCPadAir_2(UPad, phiPad, AFlr, CO2Out, CO2Air)

    # Calculate MCAirCan
    LaI = LAI
    p = P(CO2Air, LaI)
    R = 0
    CBuf = 0
    CMax_Buf = 4000
    MC_AirCan = MCAirCan(p, R, CBuf, CMax_Buf)

    # Calculate MCAirTop
    UThScr = U_TH_SCR
    KThScr = K_TH_SCR
    TAir = T_AIR
    TTop = T_TOP
    g = G
    pAir = P_AIR
    pTop = P_TOP
    f_ThScr = fThScr(UThScr, KThScr, TAir, TTop, g, pAir, pTop)
    MC_AirTop = MCAirTop(f_ThScr, CO2Air, CO2Top)

    # Calculte MCAirOut
    cleakage = C_LEAKAGE
    vWind = V_WIND
    f_leakage = fleakage(cleakage, vWind)

    Cd = CD
    URoof = U_ROOF
    USide = U_SIDE
    ARoof = A_ROOF
    ASide = A_SIDE
    hSideRoof = H_SIDE_ROOF
    TOut = T_OUT
```

```
    Cw = CW
    f_VentRoofSide = fVentRoofSide(Cd, AFlr, URoof, USide, ARoof, ASide, g,
    hSideRoof, TAir, TOut, Cw, vWind)

    ppf_VentSide = ppfVentSide(Cd, USide, ASide, vWind, AFlr, Cw)

    nSide = N_SIDE
    nSide_Thr = N_SIDE_THR
    sInsScr = S_INS_SCR
    n_InsScr = nInsScr(sInsScr)
    f_VentSide = fVentSide(n_InsScr, ppf_VentSide, f_leakage, UThScr,
    f_VentRoofSide, nSide, nSide_Thr)

    UVentForced = U_VENT_FORCED
    phiVentForced = O_VENT_FORCED
    f_VentForced = float(fVentForced(n_InsScr, UVentForced, phiVentForced, AFlr))

    MC_AirOut = MCAirOut(f_VentSide, f_VentForced, CO2Air, CO2Out)

    capCO2Air = CAP_CO2_AIR

    return (MC_BlowAir + MC_ExtAir + MCPadAir - MC_AirCan - MC_AirTop - MC_AirOut)
     / capCO2Air


# formula 2
def dxCO2Top(CO2Air, CO2Top):

    # Calculate MCAirTop
    UThScr = U_TH_SCR
    KThScr = K_TH_SCR
    TAir = T_AIR
    TTop = T_TOP
    g = G
    pAir = P_AIR
    pTop = P_TOP
    f_ThScr = fThScr(UThScr, KThScr, TAir, TTop, g, pAir, pTop)
    MC_AirTop = MCAirTop(f_ThScr, CO2Air, CO2Top)

    # Calculate MCTopOut
    AFlr = A_FLR
    Cd = CD
    URoof = U_ROOF
    USide = U_SIDE
    ARoof = A_ROOF
    ASide = A_SIDE
    hSideRoof = H_SIDE_ROOF
    TOut = T_OUT
    Cw = CW
    vWind = V_WIND
    f_VentRoofSide = fVentRoofSide(Cd, AFlr, URoof, USide, ARoof, ASide, g,
    hSideRoof, TAir, TOut, Cw, vWind)


    hVent = H_ROOF
    ppf_VentRoof = ppfVentRoof(Cd, URoof, ARoof, AFlr, g, hVent, TAir, TOut, Cw,
    vWind)


    cleakage = C_LEAKAGE
    f_leakage = fleakage(cleakage, vWind)

```

```
    sInsScr = S_INS_SCR
    n_InsScr = nInsScr(sInsScr)
    nRoof = N_ROOF
    nSide = N_SIDE
    nRoof_Thr = N_ROOF_THR
    f_VentRoof = fVentRoof(n_InsScr, f_leakage, UThScr, f_VentRoofSide, nRoof,
    nSide, nRoof_Thr, ppf_VentRoof)

    CO2Out = CO2_OUT
    MC_TopOut = MCTopOut(f_VentRoof, CO2Top, CO2Out)

    capCO2Top = CAP_CO2_TOP
    return (MC_AirTop - MC_TopOut) / capCO2Top


def dx(CO2_AIR, CO2_TOP):
    return dxCO2Air(CO2_AIR, CO2_TOP), dxCO2Top(CO2_AIR, CO2_TOP)
```

Listing 2: **dx** function

3.2 Exercise 3

When taking and analysing the data-set from GitHub, we saw that Tair, Tout, Top, Vwind, and some control coefficients U were variables that could be changed during the validating time. Assuming that those are constant and we found specific and reasonable values for them at an arbitrary time to check if the program worked correctly . In particular, we chose t = t0, took data from team AiCU in GH Cucumber and received:

Tair = 293.25 K, Tout = 290.85 K, Ttop = Tair + 1 = 294.25 K
UThScr = 0, Vwind = 3.2 (m/s), Uroof = (ventlee + ventwind)/2 = (0 + 1.2)/2 = 0.6 % = 0.006
CO2Air = CO2Top = 790.7675669 mg/m^3
CO2Out = 707.96 mg/m^3

Applying them, we run the program and print out the results of **dx** to the screen:

```
(CO2'_Air, CO2'_Top) =
(0.8925634680669298, -0.03968863092819846)

Process finished with exit code 0
```

The result showed that at t = t0,
CO2'_Air = 0.8925 (increasing)
CO2'_Top = -0.0396 (decreasing)

3.3 Exercise 4

3.3.1 Explicit Euler and Runge Kutta algorithm in Python programs

Based on explicit Euler and Runge Kutta of order 4 algorithms for solving first-order differential equations we had presented on the background, the following programs are Python code to estimate the result of the system. All 2 programs had some inputs: a callable function as dx, the values at time t0 of CO2_Air and CO2_Top, time t0, and the time step h.

The functions used 2 arrays which is t[] and f[], t[] is an array to store each time point starting at t0, f is a 2-dimensional array, which contains 2 sub-arrays storing the values of CO2 concentration Air and Top. We used **numpy** package to implement them. Moreover, the data of some variables and coefficents always changes depending on time; therefore, to get the value of them, we create a excel file called **data.xlsx** storing the value of Tair, Tout, Ttop, UThScr, Vwind and Uroof (taken from team AiCU). To do that, we need to use **pandas** package to read and get the dataframe **df**.

First program is a function named **euler**, which is used to estimate the results using explicit Euler.

```
def euler(dx, init_co2_air, init_co2_top, t0, h):
    number_of_steps = N
    t = np.zeros(number_of_steps + 1)
    f = np.zeros((number_of_steps + 1, 2))
    f[0, :] = init_co2_air, init_co2_top
    t[0] = t0
    data = pd.read_excel("data.xlsx")
    df = pd.DataFrame(data)
    i = 0
    for k in range(number_of_steps):
        global T_AIR, T_OUT, T_TOP, V_WIND, U_ROOF, U_TH_SCR, U_EXT_CO2
        if (k % 300) == 0:
            T_AIR = float(df.at[i, "Tair"])
            T_OUT = float(df.at[i, "Tout"])
            T_TOP = float(df.at[i, "Ttop"])
            V_WIND = float(df.at[i, "Vwind"])
            U_ROOF = float(df.at[i, "Uroof"])
            U_TH_SCR = float(df.at[i, "UThScr"])
            i += 1
        t[k + 1] = t[k] + h
        f[k + 1, 0] = f[k, 0] + h * dx(f[k, 0], f[k, 1])[0]
        f[k + 1, 1] = f[k, 1] + h * dx(f[k, 0], f[k, 1])[1]
    return f, t
```

Listing 3: **euler** function

Second program is a function named **rk4**, which is used to estimate the results using explicit Euler.

```
def rk4(dx, init_co2_air, init_co2_top, t0, h):
    number_of_steps = N
    t = np.zeros(number_of_steps + 1)
    f = np.zeros((number_of_steps + 1, 2))
    t[0] = t0
    f[0, :] = init_co2_air, init_co2_top
```

```
    data = pd.read_excel("data.xlsx")
    df = pd.DataFrame(data)
    i = 0
    for k in range(number_of_steps):
        t[k + 1] = t[k] + h
        if (k % 300) == 0:
            global T_AIR, T_OUT, T_TOP, V_WIND, U_ROOF, U_TH_SCR
            T_AIR = float(df.at[i, "Tair"])
            T_OUT = float(df.at[i, "Tout"])
            T_TOP = float(df.at[i, "Ttop"])
            V_WIND = float(df.at[i, "Vwind"])
            U_ROOF = float(df.at[i, "Uroof"])
            U_TH_SCR = float(df.at[i, "UThScr"])
            i += 1
        k1_0 = dx(f[k, 0], f[k, 1])[0]
        k1_1 = dx(f[k, 0], f[k, 1])[1]
        k2_0 = dx(f[k, 0] + 0.5 * h * k1_0, f[k, 1] + 0.5 * h * k1_0)[0]
        k2_1 = dx(f[k, 0] + 0.5 * h * k1_1, f[k, 1] + 0.5 * h * k1_1)[1]
        k3_0 = dx(f[k, 0] + 0.5 * h * k2_0, f[k, 1] + 0.5 * h * k2_0)[0]
        k3_1 = dx(f[k, 0] + 0.5 * h * k2_1, f[k, 1] + 0.5 * h * k2_1)[1]
        k4_0 = dx(f[k, 0] + h * k3_0, f[k, 1] + h * k3_0)[0]
        k4_1 = dx(f[k, 0] + h * k3_1, f[k, 1] + h * k3_1)[1]
        f[k + 1, 0] = f[k, 0] + (1.0/6.0) * h * (k1_0 + 2 * (k2_0 + k3_0) + k4_0)
        f[k + 1, 1] = f[k, 1] + (1.0/6.0) * h * (k1_1 + 2 * (k2_1 + k3_1) + k4_1)
    return f, t
```

Listing 4: **rk4** function

These solvers returned 2 arrays f[] and t[], which stored all data during each calculation step. Therefore, to get approximate values of CO2 (Air and Top) at some time t', we could access to f[i, :] (i is i-th step).

The data we took from team AiCU has an interval of 5 minutes between 2 points. In the next task, we are required to calculate and compare with the actual data, thus we chose h = 1 (s), which is quite small enough to estimate. Then the coefficents and variables had to be updated each 300s when we run the loop.

3.3.2 Find the approximate values and calculate the difference of the result

In this task, we will try to run the program with specific values of CO2 Air and CO2 Top at time t from the data set <AiCU> as initial values. We will give CO2_Air = CO2_Top = 790.7675669, time t0 = 0, h = 1s and N = 600000 (1 week), run the program and get 2 returned arrays f[] and t[].
Then, we made a few steps in **main** function to get the data table in 5 minutes, 10 minutes, 20 minutes, ...till 120 minutes
The following result was made by using euler method:

Time	CO2_Air	CO2_Top	CO2_Air(actual
0	790.7676	790.7676	790.7675669
300	885.719	853.1349	799.7945946
600	918.3009	880.6296	774.5189203
900	930.1398	890.7074	794.3783757
1200	934.3449	894.261	785.351352
1500	935.8377	895.5226	788.962162
1800	936.3689	895.9718	783.5459466
2100	955.2809	916.661	803.4054072
2400	963.9	924.2803	821.4594576
2700	966.9529	926.8135	794.3783793
3000	968.186	927.8804	799.7945946
3300	968.922	928.5726	792.5729726
3600	948.9041	906.6916	787.1567575
3900	941.3455	900.259	801.599999
4200	938.8165	898.161	805.2108107
4500	921.4593	879.5809	799.7945952
4800	915.7129	874.8021	817.8486472
5100	913.9114	873.3132	803.4054069
5400	913.234	872.7239	821.4594589
5700	912.9078	872.425	807.0162162
6000	912.9184	872.4634	814.2378365
6300	912.6975	872.2221	794.378381
6600	912.7384	872.2853	823.2648629
6900	912.8654	872.4196	808.8216221

time (s), CO2_Air, Top, Air(actual) (mg/m^3)

The mean absolute error is $\sum_{n=1}^{4} |CO2\_Air(actual) - CO2\_Air|/n = 124.9628152$.

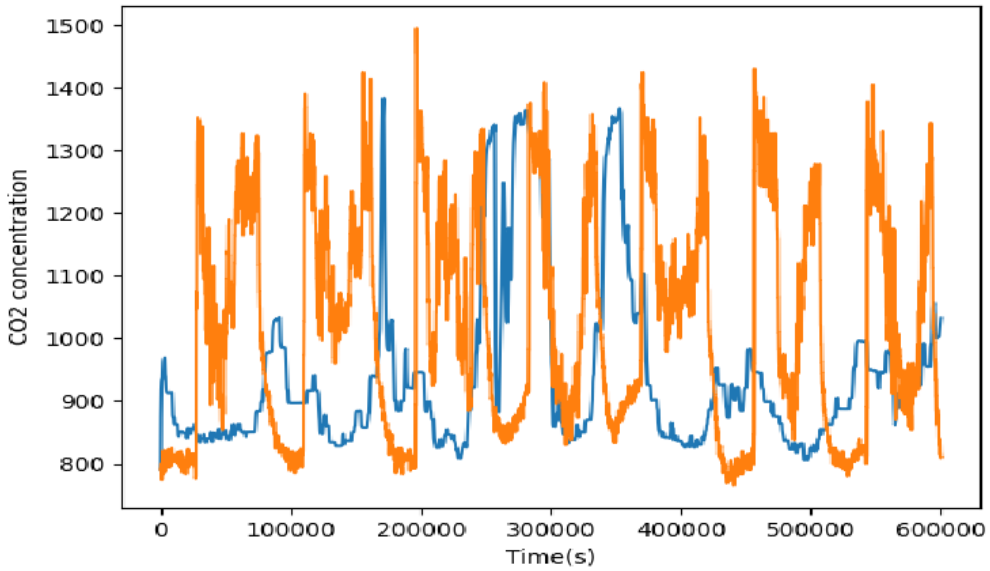
The following result was made by using rk4 method:

Time	CO2_Air	CO2_Top	CO2_Air(actual)	Difference	Mean error
0	790.7676	790.7676	790.7675669	1.727E-09	124.95696
300	885.6515	853.0662	799.7945946	85.856939	
600	918.2491	880.5816	774.5189203	143.73018	
900	930.1112	890.6817	794.3783757	135.73285	
1200	934.3312	894.2488	785.351352	148.97983	
1500	935.8315	895.5172	788.962162	146.86936	
1800	936.3662	895.9695	783.5459466	152.82026	
2100	955.2618	916.642	803.4054072	151.8564	
2400	963.885	924.2663	821.4594576	142.42553	
2700	966.9444	926.8057	794.3783793	172.56602	
3000	968.1815	927.8764	799.7945946	168.38692	
3300	968.9196	928.5704	792.5729726	176.34662	
3600	948.9233	906.7105	787.1567575	161.76657	
3900	941.3588	900.2712	801.599999	139.75878	
4200	938.8234	898.1672	805.2108107	133.61261	
4500	921.4792	879.5997	799.7945952	121.68459	
4800	915.7242	874.8122	817.8486472	97.875511	
5100	913.9165	873.3177	803.4054069	110.5111	
5400	913.2362	872.7258	821.4594589	91.776772	
5700	912.9088	872.4259	807.0162162	105.89262	
6000	912.9187	872.4636	814.2378365	98.680824	
6300	912.6978	872.2224	794.378381	118.31941	
6600	912.7385	872.2853	823.2648629	89.473622	
6900	912.8653	872.4195	808.8216221	104.04369	

The mean absolute error is $\sum_{n=1}^{4} |CO2\_Air(actual) - CO2\_Air|/n = 124.9569576$

In conclusion, the result we received when calculate using rk4 gived less error than euler. However, both 2 methods need an extremely small step size (h) to make the result come nearly to the actual data.

We had plot a graph to show the difference between estimated data and actual data. The following graph presents for euler(blue) and actual(orange).



The actual data and our result showed a large difference. This can be caused by our invalid data taken from source or large error(local error) in choosing step size h when calculates.

3.4 Exercise 5

3.4.1 Restate the VP(vapor pressure) model

a. **State variables of the model**
As described in section 2, thermal screen divides a greenhouse into two different compartments, above and below thermal screen affect with tempreture has the same effect as CO2 gas . State variables of the model which affect the temprature are described in the figure below.

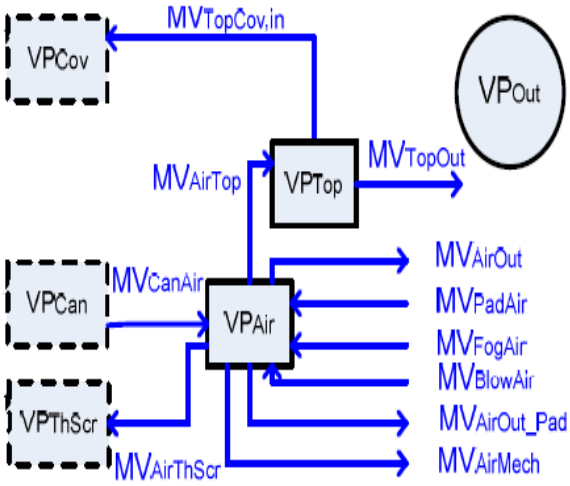


figure1

b. **Vapour pressure of the greenhouse air and the air in the top compartment**
The vapour pressure of the greenhouse air VP_{Air} is described by:

$$cap_{VPair}VP_{air} = MV_{CanAir} + MV_{PadAir} + MV_{FogAir} + MV_{BlowAir} - MV_{AirMech} \\ -MV_{AirThScr} - MV_{AirOut} - MV_{AirTop} - MV_{AirOut\_Pad} \quad (1)$$

where cap_{VPair} is the capacity of the air to store water vapour. Vapour is exchanged between the air and surrounding elements i.e. the canopy MV_{CanAir}, the outlet air of the pad MV_{PadAir}, the fogging system MV_{FogAir}, the direct air heater $MV_{BlowAir}$, the thermal screen $MV_{AirThScr}$, the top compartment air MV_{AirOut}, the outdoor air due to the air exchange caused by the pad and fan system MV_{AirOut_Pad} and the mechanical cooling system $MV_{AirMech}$.
The vapour pressure of the air in the top compartment VPTop is described by:

$$cap_{VPtop}VP_{Top} = MV_{AirTop} - MV_{TopCov,in} - MV_{TopOut} \quad (2)$$

where cap_{VPtop} is the capacity of the top compartment to store water vapour, $MV_{TopCov,in}$,in is the vapour exchange between the top compartment and the internal cover layer and MV_{TopOut} is the vapour exchange between the top compartment and the outside air.
Note that in formulas (1) and (2), some assumptions have been decided such as the vapour pressure of the greenhouse air and the air in the top compartment is not affected by any other sources except those shown in figure 1. Beside, the greenhouse is a perfectly environmental sense that the steam pressure is even distributed in the lower and upper spaces. The notations cap_A, VP_A and MV_AB are respectively the capacity to store vapor in A(m), the vapor pressure in A (kgm^2s^{-1}) and the net vapor flux from A to B (kgm^2s^1),where Air and Top represent the lower and upper compartments, Blow represents the direct air heater, Ext represents the source from the third party, Pad represents the pad system ,Can represents the total foliage of the plants inside the greenhouse, and Out represents the space outside the greenhouse.
Here are formulas to calculate MC_AB. First, we consider the amount of vapor going from the heater into the greenhouse air as follows:

$$MV_{BlowAir} = \eta_{HeatVap}H_{BlowAir} = \frac{\eta_{HeatVap}U_{Blow}H_{Blow}}{A_{Flr}}$$

In this formula , $\eta_{HeatVap}$ is the amount of vapor generated when 1 Joule of sensible heat is generated by the heater ($mgvaporJ^{-1}$).
Similarly, the amount of CO2that is pumped into the greenhouse by the third party that supplies CO2is given by:

$$MV_{FogAir} = \frac{U_{Fog}\phi_{Fog}}{A_{Flr}}$$

The notations U_Fog and ϕ_{Fog} are respectively a dimensionless parameter in [0,1] that adjusts the rate at which the vapor is injected into the greenhouse and that represents the third party's ability to pump vapour (mgs^{-1}).
On the other hand, the amount of vapour that enters the greenhouse through the pad system is due to the difference in the concentration of vapour inside and outside the greenhouse and the ability of the pad system for the air to go through. Furthermore, the pad can be adjusted to let in more air. The following formula is used to calculate MV_PadAir:

$$MV_{PadAir} = \rho_{Air} f_{Pad}(\eta_{Pad}(x_{Pad} - x_{Out}) + x_{Out}) = \frac{1}{A_{Flr}} U_{Pad}\phi_{Pad}\rho_{Pad}(\eta_{Pad}(x_{Pad} - x_{Out}) + x_{Out})$$

The flux $f_{Pad}(ms^1)$ is the product of the dimensionless parameter U_Pad in [0,1], which represents the permeability of the pad and ϕ_{Pad}, which is the ability for the vapor to pass through (m^3s^1) divided by the area of the greenhouse floor.
The net flux of vapor from the lower compartment to the upper compartment of the greenhouse is more complicated and it depends on the difference in temperature and air density between the two compartments.

$$MV_{AirOut\_Pad} = f_{Pad}\frac{M_{Water}}{R}\frac{VP_{Air}}{T_Air} = \frac{1}{A_{Flr}} U_{Pad}\phi_{Pad}\frac{M_{Water}}{R}\frac{VP_{Air}}{T_Air}[kgm^{-2}s^{-1}]$$

These variable $MV_{AirThScr}, MV_{TopCov,in}, MV_{AirMech}$ is caculated by the follow formula:

$$MV_{12} = \frac{1}{1 + exp(S_{MV12}(VP_1 - VP_2))} 6.4.10^{-9} HEC_{12}(VP_1 - VP_2)$$

where S_{MV12} (-) is the slope of the differentiable switch function for vapour pressure differences, $HEC_{12}(Wm^{-2}K^{-1})$ is the heat exchange coefficient between object 1 and 2.
The heat exchange coefficient between the surface of the mechanical cooling unit and the greenhouse air is determined by:

$$HEC_{MechAir} = \frac{(U_{MechCool}COP_{MechCool}P_{MechCool}/A_{Flr})}{T_{Air}T_{Mech} + 6.4.10^{-9} \triangle H(VP_{Air} - VPMechCool)}$$

Where $U_{MechCool}(-)$ is the control valve of the mechanical cooling mechanism, $COP_{MechCool}(-)$ is the coefficient of performance of the mechanical cooling system and $P_{MechCool}(W)$ is the electrical capacity of the mechanical cooling system, $T_{MechCool}(^oK)$ is the temperature of the cooling surface which is an input of the model, and $VP_{MechCool}(Pa)$ is the saturated vapour pressure of the mechanical cooling mechanism.

$$HEC_{TopCov,in} = c_{HECin}(T_{Top} - T_{Cov,in})^{0.33}\frac{A_{Cov}}{A_{Flr}} HEC_{AirThScr} = 1.7U_{ThScr}|T_{Air} - T_{ThScr}|^{0.33}$$

The vapour flux from the greenhouse air compartment to the thermal screen, $MV_{AirThScr}$ and the vapour flux from the top compartment to the internal cover layer $MV_{TopCov,in}$, are described as below.

$$MV_{12} = \frac{M_{Water}}{R} f_{12}(\frac{VP_1}{T_1} - \frac{VP_2}{T_2})$$

where MV_{12} is the vapour flux from location 1 to location 2, $f_12(m^3m^{-2}s^{-1})$ is the air flux from location 1 to location 2, $T_1(^oK)$ is the temperature at location 1 and $T_2(^oK)$ is the temperature at location 2.
The vapour fluxes MV_{AirTop}, MV_{AirOut} and MV_{TopOut} are described analogously where by their accompanying air fluxes are f_{ThScr} (the flux through the thermal screen), $f_{VentForced} + f_{VentSide}$(the flux due to natural ventilation through the side windows or forced ventilation) and $f_{VentRoof}$ (flux due to roof ventilation) respectively.
These variables f_{ThScr}, $f_{VentSide}$, $f_{VentRoof}$, $f_{VentForced}$ are simillar as question 2.
The canopy transpiration is described by:

$$MV_{CanAir} = VEC_{CanAir}(VP_{Can} - VP_{Air})$$

where $VEC_{CanAir}(kgPas^{-1})$ is the vapour exchange coefficient between the canopy and air, VP_{Can} is the saturated vapour pressure at canopy temperature. According to Stanghellini (1987) the vapour transfer coefficient of the canopy transpiration can be calculated by:

$$VEC_{CanAir} = \frac{2\rho_{Air}c_{p,Air}LAI}{\triangle H\Upsilon(r_b + r_s)}$$

where $\rho_{Air}(kgm^{-3})$ is the density of the greenhouse air, $c_{p,Air}(JK^{-1}kg^{-1})$ is the specific heat capacity of the greenhouse air, $LAI(m^2m^{-2})$ is the leaf area index, $\triangle H(Jkg^{-1})$ is the latent heat of evaporation of water, $\Upsilon(PaK^{-1})$ is the psychometric constant, $r_b(sm^{-1})$ is the boundary layer resistance of the canopy for vapour transport and $r_s(sm^{-1})$ is the stomatal resistance of the canopy for vapour transport.

3.4.2 Calculation

We had build a program to calculate the net VP flux from one place to an other place, the following Python code is build to present the functions:

```
# ----------Cap----------

def Cap_VP_Air(M_Water, H_Air, R, T_Air):
    return (M_Water * H_Air) / (R * T_Air)


def Cap_VP_Top(M_Water, H_Top, R, T_Top):
    return (M_Water * H_Top) / (R * T_Top)


# ----------MV_BlowAir----------

def MV_BlowAir(N_Heat_Vap, U_Blow, P_Blow, A_Flr):
    return (N_Heat_Vap * U_Blow * P_Blow) / A_Flr


# ----------MV_FogAir----------

def MV_FogAir(U_Fog, O_Fog, A_Flr):
    return (U_Fog * O_Fog) / A_Flr


# ----------MV_PadAir----------

def MV_PadAir(U_Pad, O_Pad, P_Air, N_Pad, X_Pad, X_Out, A_Flr):
```

```
    return (U_Pad * O_Pad * P_Air * (N_Pad * (X_Pad - X_Out) + X_Out)) / A_Flr


# ----------MV_AirOut_Pad----------

def MV_AirOut_Pad(U_Pad, O_Pad, A_Flr, M_Water, R, VP_Air, T_Air):
    return (U_Pad * O_Pad * M_Water * VP_Air) / A_Flr * R * T_Air


# ----------MV_Air_ThScr----------

def HEC_Air_ThScr(U_ThScr, T_ThScr, T_Air):
    return 1.7 * U_ThScr * pow(abs(T_Air - T_ThScr), 0.33)


def MV_Air_ThScr(S_MV_Air_ThScr, VP_Air, VP_ThScr, HEC_Air_ThScr):
    return (6.4 * pow(10, -9) * HEC_Air_ThScr * (VP_Air - VP_ThScr)) / (
                1 + pow(2.718, S_MV_Air_ThScr * (VP_Air - VP_ThScr)))


# ----------MV_Top_Cov_in----------

def HEC_Top_Cov_in(c_Hec_in, T_Top, T_Cov_in, A_Cov, A_Flr):
    return c_Hec_in * pow(abs(T_Top - T_Cov_in), 0.33) * A_Cov / A_Flr


def MV_Top_Cov_in(S_MV_Top_Cov_in, VP_Top, VP_Cov_in, HEC_Top_Cov_in):
    a = 1 + pow(2.718, S_MV_Top_Cov_in * (VP_Top - VP_Cov_in))
    return (6.4 * pow(10, -9) * HEC_Top_Cov_in * (VP_Top - VP_Cov_in)) / a

# ----------MV_AirMech----------

def HEC_AirMech(U_MechCool, COP_MechCool, P_MechCool, A_Flr, T_Air, T_Mech,
    Delta_H, VP_Air, VP_MechCool):
    a = U_MechCool * COP_MechCool * P_MechCool / A_Flr
    b = T_Air - T_Mech + 6.4 * pow(10, -9) * Delta_H * (VP_Air - VP_MechCool)
    return a / b


def MV_Air_Mech(S_MV_Air_Mech, VP_Mech, VP_Air, HEC_Air_Mech):
    return (6.4 * pow(10, -9) * HEC_Air_Mech * (VP_Air - VP_Mech)) / (
                1 + pow(2.718, S_MV_Air_Mech * (VP_Air - VP_Mech)))


# ----------f-Formula----------

def f_ThScr(U_ThScr, K_ThScr, T_Air, T_Top, g, p_Air, p_Top):
    a = U_ThScr * K_ThScr * pow(abs(T_Air - T_Top), 2 / 3)
    PMean_Air = (p_Air + p_Top) / 2
    b = (1 - U_ThScr) * pow(g * (1 - U_ThScr) * abs(p_Air - p_Top) / (2 *
    PMean_Air), 1 / 2)
    return a + b


def fleakage(cleakage, vWind):
    if vWind < 0.25:
        return 0.25 * cleakage
    else:
        return vWind * cleakage


def nInsScr(sInsScr):
```

```
    return sInsScr * (2 - sInsScr)


def f_Vent_Roof_Side(Cd, AFlr, URoof, USide, ARoof, ASide, g, hSideRoof, TAir, TOut, Cw, vWind):
    a = Cd / AFlr
    b = pow(URoof * USide * ARoof * ASide, 2) / (pow(URoof * ARoof, 2) + pow(USide * ASide, 2))
    TMean_Air = (TAir + TOut) / 2
    c = 2 * g * hSideRoof * (TAir - TOut) / TMean_Air
    _d = (URoof * ARoof + USide * ASide) / 2
    d = pow(_d, 2) * Cw * pow(vWind, 2)
    return a * sqrt(b * c + d)


def ppfVentSide(Cd, USide, ASide, vWind, AFlr, Cw):
    return Cd * USide * ASide * vWind * sqrt(Cw) / (2 * AFlr)


def f_VentSide(n_InsScr, ppf_VentSide, f_leakage, UThScr, f_VentRoofSide, nSide, nSide_Thr):
    if nSide >= nSide_Thr:
        return n_InsScr * ppf_VentSide + 0.5 * f_leakage
    else:
        return n_InsScr * (UThScr * ppf_VentSide + (1 - UThScr) * f_VentRoofSide * nSide) + 0.5 * f_leakage


def f_VentForced(n_InsScr, U_VentForced, phi_VentForced, A_Flr):
    return n_InsScr * U_VentForced * phi_VentForced / A_Flr


def ppfVentRoof(Cd, URoof, ARoof, AFlr, g, hVent, TAir, TOut, Cw, vWind):
    TMeanAir = (TAir + TOut) / 2
    part1 = Cd * URoof * ARoof / (2 * AFlr)
    part2 = g * hVent * (TAir - TOut) / 2 / TMeanAir + Cw * pow(vWind, 2)
    return part1 * sqrt(part2)


def fVentRoof(n_InsScr, f_leakage, UThScr, ppf_VentRoofSide, nRoof, nSide, nRoof_Thr, ppf_VentRoof):
    if nRoof >= nRoof_Thr:
        return n_InsScr * ppf_VentRoof + 0.5 * f_leakage
    else:
        return n_InsScr * (UThScr * ppf_VentRoof + (1 - UThScr) * ppf_VentRoofSide * nSide) + 0.5 * f_leakage


# ----------MV_AirTop----------

def MV_AirTop(M_Water, R, f_Th_Scr, VP_Air, VP_Top, T_Air, T_Top):
    return M_Water * f_Th_Scr * (VP_Air / T_Air - VP_Top / T_Top) / R


# ----------MV_AirOut----------

def MV_AirOut(M_Water, R, VP_Air, VP_Out, T_Air, T_Out, f_vent_Side, f_vent_Forced):
    return M_Water * (f_vent_Forced + f_vent_Side) * (VP_Air / T_Air - VP_Out / T_Out) / R


```

```
# ————————————MV_TopOut————————————

def MV_TopOut(M_Water, R, VP_Top, VP_Out, T_Top, T_Out, f_vent_roof):
    return M_Water * f_vent_roof * (VP_Top / T_Top - VP_Out / T_Out) / R


# ————————————MV_CanAir————————————

def VEC_CanAir(p_Air, c_p_Air, LAI, Delta_H, y, r_b, r_s):
    return (2 * p_Air * c_p_Air * LAI) / (Delta_H * y * (r_s + r_b))


def MV_CanAir(VEC_CanAir, VP_Can, VP_Air):
    return VEC_CanAir * (VP_Can - VP_Air)
```

Listing 5: Vapour flux Functions

Using these functions and some available coefficients, variables, we had build a function named dx with 2 parameters(VP_{Air} and VP_{Top}). These parameters are respectively vapour pressure of the greenhouse air in the lower and upper compartments of the green house. The function dx presents for a dynamical system:

The following Python code is written to present for **dx** function

```
def dx(VP_Air, VP_Top):
    # MV_BlowAir
    mv_blow_air = MV_BlowAir(N_HEAT_VAP, U_BLOW, P_BLOW, A_FLR)

    # MV_FogAir
    mv_fog_air = MV_FogAir(U_FOG, O_FOG, A_FLR)

    # MV_PadAir
    mv_pad_air = MV_PadAir(U_PAD, O_PAD, P_AIR, N_PAD, X_PAD, X_OUT, A_FLR)

    # MV_AirOut_Pad
    mv_airout_pad = MV_AirOut_Pad(U_PAD, O_PAD, A_FLR, M_WATER, R, VP_Air, T_AIR)

    # MV_Air_ThScr
    hec_air_thscr = HEC_Air_ThScr(U_THSCR, T_THSCR, T_AIR)
    mv_air_thscr = MV_Air_ThScr(S_MV_12, VP_Air, VP_THSCR, hec_air_thscr)

    # MV_Top_Cov_in
    hec_top_cov_in = HEC_Top_Cov_in(C_HEC_IN, T_TOP, T_COV_IN, A_COV, A_FLR)
    mv_top_cov_in = MV_Top_Cov_in(S_MV_12, VP_Top, VP_COV_IN, hec_top_cov_in)

    # MV_AirMech
    hec_air_mech = HEC_AirMech(U_MECH_COOL, COP_MECHCOOL, P_MECHCOOL, A_FLR, T_AIR
    , T_MECH, DELTA_H, VP_Air, VP_MECH)
    mv_air_mech = MV_Air_Mech(S_MV_12, VP_MECH, VP_Air, hec_air_mech)

    # MV_CanAir
    vec_can_air = VEC_CanAir(P_AIR, CP_AIR, LAI, DELTA_H, Y, R_B, R_S)
    mv_can_air = MV_CanAir(vec_can_air, VP_CAN, VP_Air)

    # f-Formula
    n_ins_scr = nInsScr(S_INS_SCR)
    f_th_scr = f_ThScr(U_THSCR, K_THSCR, T_AIR, T_TOP, G, P_AIR, P_TOP)
    f_leakage = fleakage(C_LEAKAGE, V_WIND)
    f_vent_roof_side = f_Vent_Roof_Side(CD, A_FLR, U_ROOF, U_SIDE, A_ROOF, A_SIDE,
     G, H_SIDE_ROOF, T_AIR, T_OUT, CW,
                                        V_WIND)
```

```
    ppf_vent_side = ppfVentSide(CD, U_SIDE, A_SIDE, V_WIND, A_FLR, CW)
    f_vent_side = f_VentSide(n_ins_scr, ppf_vent_side, f_leakage, U_THSCR,
    f_vent_roof_side, N_SIDE, N_SIDE_TH)
    f_vent_forced = f_VentForced(n_ins_scr, U_VENTFORCED, O_VENTFORCED, A_FLR)
    ppf_vent_roof = ppfVentRoof(CD, U_ROOF, A_ROOF, A_FLR, G, H_VENT, T_AIR, T_OUT
    , CW, V_WIND)
    f_vent_roof = fVentRoof(n_ins_scr, f_leakage, U_THSCR, f_vent_roof_side,
    N_ROOF, N_SIDE, N_ROOF_TH, ppf_vent_roof)

    # MV_AirTop
    mv_air_top = MV_AirTop(M_WATER, R, f_th_scr, VP_Air, VP_Top, T_AIR, T_TOP)

    # MV_AirOut
    mv_air_out = MV_AirOut(M_WATER, R, VP_Air, VP_OUT, T_AIR, T_OUT, f_vent_side,
    f_vent_forced)

    # MV_TopOut
    mv_top_out = MV_TopOut(M_WATER, R, VP_Top, VP_OUT, T_TOP, T_OUT, f_vent_roof)

    # dx
    cap_VP_air = Cap_VP_Air(M_WATER, H_AIR, R, T_AIR)
    cap_VP_top = Cap_VP_Top(M_WATER, H_TOP, R, T_TOP)
    dx_VP_air = (mv_can_air + mv_pad_air + mv_fog_air + mv_blow_air - mv_air_mech
    - mv_air_thscr
                 - mv_air_out - mv_air_top - mv_airout_pad) / cap_VP_air
    dx_VP_top = (mv_air_top - mv_top_cov_in - mv_top_out) / cap_VP_top
    return dx_VP_air, dx_VP_top
```

Listing 6: **dx** function

3.4.3 Validate with specific data

All values for each input parameter of function dx including the difference in temperature and in air density are listed below:

Latent heat of evaporation: $\triangle H = 2.45.10^6$ (J kg^{-1} water).

Amount of CO_2 which is released when 1 Joule of sensible energy is produced by the heat blower: $\eta_{HeatCO_2} = 0.557$ (kg vapour J^{-1}).

The control valve of the direct air heater ranging in [0, 1]: $U_{Blow} = 0.5$

The heat capacity of the direct air heater: $P_{Blow} = 0.5.10^6$ (W)

Surface of the greenhouse floor: $A_{Flr} = 1.4.10^4$ (m^2)

Surface of the cover including side-walls: $A_{Cov} = 1.8.10^4$ (m^2)

The control valve of the fogging system ranging in [0, 1]: $U_{Fog} = 0.5$

The capacity of the fogging system: $\phi_{Fog} = 0$ (kg water s^{-1})

The density of the greenhouse air below the thermal screen: $\rho_{Air} = 1.125$ (kg m^{-3})

The control valve of the pad and fan system ranging in [0, 1]: $U_{Pad} = 0.5$

The capacity of the air flux through the pad: $\phi_{Pad} = 1.67$ $(m^3 s^{-1})$

Efficiency of the pad and fan system: $\eta_{Pad} = 0$ (-)

The gravitational acceleration: g = 9.81 (m s^{-2})

The water vapour content of the pad: $x_{Pad} = 0$ (kg water kg^{-1} air)

The water vapour content of the outdoor air: $x_{Out} = 0$ (kg water kg^{-1} air)

Molar mass of water: $M_{Water} = 18$ (kg $kmol^{-1}$)

Molar gas constant: R = $8.314.10^{-3}$ (J $kmol^{-1}$ K^{-1})

The temperature below the thermal screen: $T_{Air} = 293.25$ (K)

The temperature above the thermal screen: $T_{Top} = 294.25$ (K)

The temperature of the cooling surface: $T_{Mech} = 0$ (K)

The internal cover temperature: $T_{Cov,in} = 297$ (K)

The temperature outside the greenhouse: $T_{Out} = 290.85$ (K)

The control of the roof openings ranging in [0,1]: $U_{Roof} = 0.006$

The control of the side openings ranging in [0,1]: $U_{Side} = 0.5$

The control of the forced ventilation ranging in [0,1]: $U_{VentForced} = 0.5$

The air flow capacity of the forced ventilation system: $\phi_{VentForced} = 0$ $(m^3 s^{-1})$

The slope of the differentiable switch function for vapour pressure differences: $S_{MV12} = -0.1$ (Pa^{-1})

The saturated vapour pressure of the thermal screen: $VP_{ThScr} = 2939.2$ $(kgm^2 s^{-1})$

The saturated vapour pressure of the internal cover: $VP_{Cov,in} = 2939.2$ $(kgm^2 s^{-1})$

The saturated vapour pressure of the mechanical cooling mechanism: $VP_{Mech} = 0$ $(kgm^2 s^{-1})$

The vapour partial pressure of the air outside the greenhouse: $VP_{Out} = 998.73$ $(kgm^2 s^{-1})$

The saturated vapour pressure at canopy temperature: $VP_{Can} = 2767.7$ $(kgm^2 s^{-1})$

Convective heat exchange parameter between cover and outdoor air that depends on the greenhouse shape: $c_{HECin} = 1.86$ (W m^{-2} K^{-1})

Coefficient of performance of the mechanical cooling system: $COP_{MechCool} = 0$

Electrical capacity of the mechanical cooling system: $P_{MechCool} = 0$ (W)

Specific heat capacity of the air: $c_{p,Air} = 10^3$ (J K kg^{-1})

The leaf area index: LAI = 2 ($m^2 m^{-2}$)

The psychrometric constant: $\Upsilon = 65.8$ (Pa K^{-1})

The boundary layer resistance of the canopy for vapour transport: $r_b = 275$ (s m^{-1})

The stomatal resistance of the canopy for vapour transport: $r_s = 82$ (s m^{-1})

Reduction factor: $\eta_{InsScr} = 1$ (-)

The density of the greenhouse air above the thermal screen: $\rho_{Top} = 1.52$ (kg m^{-3})

The screen flux coefficient determining the permeability of the screen: $K_{ThScr} = 0.05.10^{-3}$ (m $K^{\frac{-2}{3}}$ s^{-1})

The leakage coefficient depending on the greenhouse type: $c_{leakage} = 10^{-4}$ (-)

Wind speed: $v_{Wind} = 3.2$ (m s^{-1})

Discharge wind pressure coefficient depending on the greenhouse shape and the use of an outdoor thermal screen: $C_d = 0.75$ (-)

Global wind pressure coefficient depending on the greenhouse shape and the use of an outdoor thermal screen: $C_w = 0.09$ (-)

The roof opening area: $A_{Roof} = 0.14.10^4$ (m^2)

The side opening area: $A_{Side} = 0$ (m^2)

The vertical distance between mid-points of side wall and roof ventilation openings: $h_{SideRoof} = 2$ (m)

The ratio between the side vents area and total ventilation area: $\eta_{Side} = 1.1$ (-)

The threshold value above which no chimney effect is assumed to occur:$\eta_{Side\_Thr} = 0$ (-)

The vertical dimension of a single ventilation opening: $h_{Vent} = 0.68$ (m)

The ratio of the roof opening area: $\eta_{Roof} = 1.3$ (-)

The total ventilation area exceeds the Stack-effect threshold: $\eta_{Roof\_Thr} = 0.9$ (-)

Height of the greenhouse compartment below the thermal screen: $h_{Air} = 3.8$ (m)

Height of the greenhouse compartment above the thermal screen: $h_{Air} = 0.4$ (m)

When taking and analysing the data-set from GitHub, we saw that Tair, Tout, Top, Vwind, and some control coefficients U were variables that could be changed during the validating time. Assuming that those are constant and we found specific and reasonable values for them at an arbitrary time to check if the program worked correctly . In particular, we chose t = t0, took data from team AiCU in GH Cucumber and received:

Tair = 293.25 K, Tout = 290.85 K, Ttop = Tair + 1 = 294.25 K
UThScr = 0, Vwind = 3.2 (m/s), Uroof = (ventlee + ventwind)/2 = (0 + 1.2)/2 = 0.6 % = 0.006
VPAir = 1921.986955 kg/m^2s^{-1}
VPTop = 998.725 kg/m^2s^{-1}
VPOUT = 998.73 kg/m^2s^{-1}

Applying them, we run the program and print out the results of **dx** to the screen:

```
(VP'_Air, VP'_Top) =
(-252.52372501920442, 2431.3344818059895)

Process finished with exit code 0
```

The result showed that at t = t0,
VP'_Air = -252.52 (decreasing)
VP'_Top = 2431.33 (increasing)

3.4.4 Explicit Euler and Runge Kutta algorithm in Python programs

Based on explicit Euler and Runge Kutta of order 4 algorithms for solving first-order differential equations we had presented on the background, the following programs are Python code to estimate the result of the system. All 2 programs had some inputs: a callable function as dx, the values at time t0 of VP_{Air} and VP_{Top}, time t0, and the time step h.

The functions used 2 arrays which is t[] and f[], t[] is an array to store each time point starting at t0, f is a 2-dimensional array, which contains 2 sub-arrays storing the values of vapour pressure Air and Top. We used numpy package to implement them. Moreover, the data of some variables and coefficents always changes depending on time; therefore, to get the value of them, we create a excel file called data.xlsx storing the value of Tair, Tout, Ttop, UThScr, Vwind and Uroof (taken from team AiCU). To do that, we need to use pandas package to read and get the dataframe df. First program is a function named **euler**, which is used to estimate the results using explicit Euler.

```
1 # ----------Euler----------
```

```
def euler(dx, init_vp_air, init_vp_top, t0, h):
    number_of_steps = N
    t = np.zeros(number_of_steps + 1)
    f = np.zeros((number_of_steps + 1, 2))
    f[0, 0] = init_vp_air
    f[0, 1] = init_vp_top
    t[0] = t0
    data = pd.read_excel("dataset.xlsx")
    df = pd.DataFrame(data)
    i = 0
    for k in range(number_of_steps):
        if (k % 300) == 0:
            global T_AIR, T_OUT, T_TOP, V_WIND, U_ROOF, U_TH_SCR
            T_AIR = float(df.at[i, "Tair"])
            T_OUT = float(df.at[i, "Tout"])
            T_TOP = float(df.at[i, "Ttop"])
            V_WIND = float(df.at[i, "Vwind"])
            U_ROOF = U_TH_SCR = 0.3
            U_ROOF = float(df.at[i, "Uroof"])
            U_TH_SCR = float(df.at[i, "UThScr"])
            i += 1
        t[k + 1] = t[k] + h
        f[k + 1, 0] = f[k, 0] + h * dx(f[k, 0], f[k, 1])[0]
        f[k + 1, 1] = f[k, 1] + h * dx(f[k, 0], f[k, 1])[1]
    return f, t
```

Listing 7: **euler** function

Second program is a function named **rk4**, which is used to estimate the results using explicit Euler.

```
# ------------------Runge_Kutta----------------------

def rk4(dx, init_vp_air, init_vp_top, t0, h):
    number_of_steps = N
    t = np.zeros(number_of_steps + 1)
    f = np.zeros((number_of_steps + 1, 2))
    t[0] = t0
    f[0, 0] = init_vp_air
    f[0, 1] = init_vp_top
    data = pd.read_excel("dataset.xlsx")
    df = pd.DataFrame(data)
    i = 0
    for k in range(number_of_steps):
        t[k + 1] = t[k] + h
        if (k % 300) == 0:
            global T_AIR, T_OUT, T_TOP, V_WIND, U_ROOF, U_TH_SCR
            T_AIR = float(df.at[i, "Tair"])
            T_OUT = float(df.at[i, "Tout"])
            T_TOP = float(df.at[i, "Ttop"])
            V_WIND = float(df.at[i, "Vwind"])
            U_ROOF = U_TH_SCR = 0.3
            U_ROOF = float(df.at[i, "Uroof"])
            U_TH_SCR = float(df.at[i, "UThScr"])
            i += 1
        k1_0 = dx(f[k, 0], f[k, 1])[0]
        k1_1 = dx(f[k, 0], f[k, 1])[1]
        k2_0 = dx(f[k, 0] + 0.5 * h * k1_0, f[k, 1] + 0.5 * h * k1_0)[0]
        k2_1 = dx(f[k, 0] + 0.5 * h * k1_1, f[k, 1] + 0.5 * h * k1_1)[1]
        k3_0 = dx(f[k, 0] + 0.5 * h * k2_0, f[k, 1] + 0.5 * h * k2_0)[0]
        k3_1 = dx(f[k, 0] + 0.5 * h * k2_1, f[k, 1] + 0.5 * h * k2_1)[1]
```

```
        k4_0 = dx(f[k, 0] + h * k3_0, f[k, 1] + h * k3_0)[0]
        k4_1 = dx(f[k, 0] + h * k3_1, f[k, 1] + h * k3_1)[1]
        f[k + 1, 0] = f[k, 0] + (1.0 / 6.0) * h * (k1_0 + 2 * (k2_0 + k3_0) + k4_0
    )
        f[k + 1, 1] = f[k, 1] + (1.0 / 6.0) * h * (k1_1 + 2 * (k2_1 + k3_1) + k4_1
    )
    return f, t
```

Listing 8: **rk4** function

These solvers returned 2 arrays f[] and t[], which stored all data during each calculation step. Therefore, to get approximate values of VP (Air and Top) at some time t', we could access to f[i, :] (i is i-th step).

The data we took from team AiCU has an interval of 5 minutes between 2 points. In the next task, we are required to calculate and compare with the actual data, thus we chose h = 1 (s), which is quite small enough to estimate. Then the coefficents and variables had to be updated each 300s when we run the loop.

3.4.5 Find the approximate values and calculate the difference of the result

In this task, we will try to run the program with specific values of VP Air and VP Top at time t from the data set <AiCU> as initial values. We will give VP_Air = 2287.2592, VP_Top = 998.7252 time t0 = 0, h = 1s and N = 600000 (1 week), run the program and get 2 returned arrays f[] and t[].
Then, we made a few steps in **main** function to get the data table in 5 minutes, 10 minutes, 20 minutes, ...till 120 minutes
The following result was made by using euler method:

Time	VP_Air	VP_Top	VP_Air(actual)	Difference	Mean error
0	2287.259	998.7252	1921.986955	365.272245	118.33546
300	1955.601	1934.263	1957.702666	2.10129669	
600	1913.353	1892.868	1950.337513	36.9843827	
900	1905.58	1885.305	1947.644547	42.064213	
1200	1903.928	1883.69	1952.309564	48.3816327	
1500	1903.603	1883.372	1971.460301	67.8575887	
1800	1903.662	1883.427	1978.501232	74.8387723	
2100	1956.344	1938.659	1977.967159	21.6233029	
2400	1969.383	1951.578	2004.933125	35.5502863	
2700	1971.54	1953.628	2004.933128	33.3933435	
3000	1972.015	1954.095	1999.624021	27.6087949	
3300	1972.488	1954.596	1989.6297	17.1412222	
3600	1918.182	1897.717	2004.317972	86.1355772	
3900	1907.185	1886.922	2009.011927	101.826593	
4200	1905.342	1885.14	2001.292241	95.9503355	
4500	1856.21	1833.76	2028.548358	172.338546	
4800	1847.548	1825.29	2005.95726	158.408986	
5100	1845.775	1823.567	2023.093785	177.318624	
5400	1845.381	1823.165	2040.355974	194.974524	
5700	1845.231	1823	2042.717498	197.48662	
6000	1845.411	1823.19	2070.478758	225.067844	
6300	1845.281	1823.028	2067.2492	221.967836	
6600	1845.338	1823.102	2054.525114	209.186784	
6900	1845.43	1823.209	2072.001496	226.571767	

time (s), VP_Air, Top, Air(actual) ($kg/m^{-2}s^{-1}$)

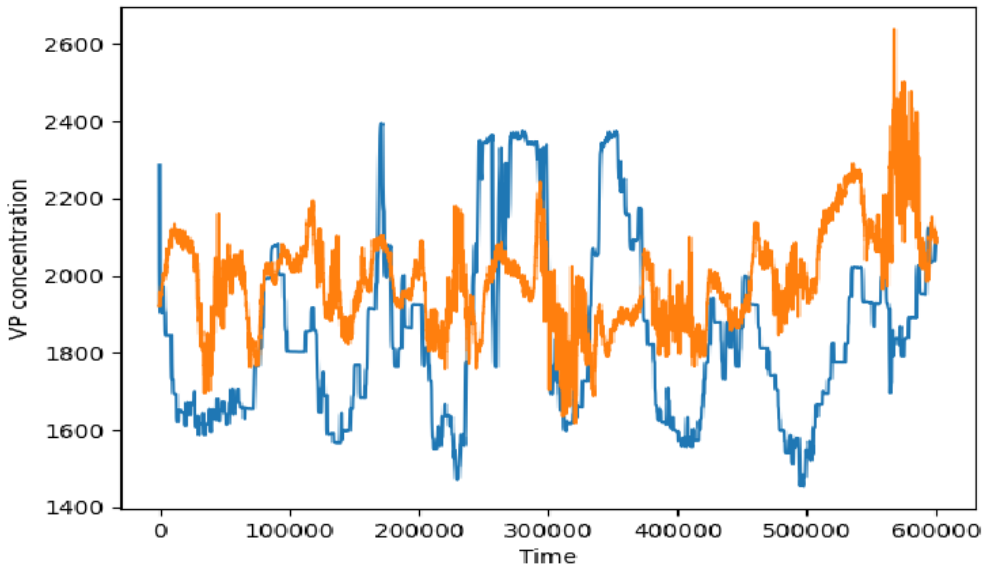
The mean absolute error is $\sum_{n=1}^{4} |VP\_Air(actual) - VP\_Air|/n = 118.33546$

The following result was made by using rk4 method:

Time	VP_Air	VP_Top	VP_Air(actual)	Difference	Mean error
0	2287.259	998.7252	1921.986955	365.272245	118.3295652
300	1955.6	1934.265	1957.702666	2.10252065	
600	1913.398	1892.913	1950.337513	36.939278	
900	1905.598	1885.322	1947.644547	42.046969	
1200	1903.933	1883.695	1952.309564	48.376451	
1500	1903.604	1883.373	1971.460301	67.8562153	
1800	1903.663	1883.428	1978.501232	74.838564	
2100	1956.287	1938.602	1977.967159	21.6806587	
2400	1969.356	1951.551	2004.933125	35.5770716	
2700	1971.532	1953.619	2004.933128	33.401585	
3000	1972.013	1954.093	1999.624021	27.6111343	
3300	1972.487	1954.595	1989.6297	17.142259	
3600	1918.241	1897.776	2004.317972	86.0765209	
3900	1907.209	1886.946	2009.011927	101.803086	
4200	1905.349	1885.146	2001.292241	95.9437021	
4500	1856.264	1833.814	2028.548358	172.284015	
4800	1847.567	1825.309	2005.95726	158.390112	
5100	1845.78	1823.572	2023.093785	177.3134	
5400	1845.383	1823.166	2040.355974	194.973177	
5700	1845.231	1823	2042.717498	197.486217	
6000	1845.411	1823.19	2070.478758	225.067969	
6300	1845.281	1823.028	2067.2492	221.96771	
6600	1845.338	1823.102	2054.525114	209.186827	
6900	1845.43	1823.209	2072.001496	226.571877	

The mean absolute error is $\sum_{n=1}^{24} |VP\_Air(actual) - VP\_Air|/n = 118.3295652$

In conclusion, the result we received when calculate using rk4 gived less error than euler. However, both 2 methods need an extremely small step size (h) to make the result come nearly to the actual data.
We had plot a graph to show the difference between estimated data and actual data. The following graph presents for **euler(blue) and actual(orange).**



The actual data and our result showed a large difference. This can be caused by our invalid data taken from source or large error(local error) in choosing step size h when calculates.

4 Conclusion

In this assignment, we had learned and carried out lots of tasks:

- Study 2 numerical methods to solve systems of first-order ODEs. (explicit Euler and Runge Kutta).
- Be able to apply them to model the dynamical systems in green house.
- Estimate and forecast the Greenhouse Micro-climate and understand how it can be modeled.

Then, we can understand more about real-life applications of algorithms and be able to apply and solve the problem.

References

[1] JOHN THOMAS, A brief overview of non-linear ordinary differential equations

[2] Systems of First Order Differential Equations

[3] A model based greenhouse design method - Vanthoor 2011

[4] Elementary Numerical Analysis 2003