Analyzing Aircraft Dynamic System Stability for Autopilot Control through Complex Eigenvalues

Naomi Risaka Sitorus – 13523122[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523122@std.stei.itb.ac.id, [2]naomi.risaka@gmail.com

***Abstract*—Autopilot is a vital control system in modern aviation, designed to stabilize an aircraft's heading and altitude, ensuring a consistent flight path. Stability in such systems is determined by the aircraft's dynamic behavior, which can be analyzed using mathematical concepts like complex eigenvalues. This study focuses on the application of complex eigenvalues in understanding aircraft dynamic system stability for autopilot control. The control matrix of the system generates complex eigenvalues. If the real parts are negative, the system is stable. Otherwise, the system is unstable as oscillations occur. This study uses oscillation theory to analyze these behaviors and provides control adjustments to achieve system stability, where oscillations are near zero. The developed program replicates an autopilot stabilization system, showing how adjustments can be made to stabilize the aircraft based on the analysis of eigenvalues. This study provides valuable insights into the practical application of complex eigenvalues, which can contribute to advancements in autopilot algorithms and other stability control fields, ultimately helping to create more reliable and efficient control systems.**

***Keywords*—Aircraft Stabilization, Autopilot Systems, Complex Numbers, Eigenvalues.**

I. Introduction

In today's aviation world, autopilot is an essential system in an aircraft that controls the aircraft without continuous direct human assistance. This system is capable of performing many functions, ranging from maintaining a constant heading and altitude during cruising and stabilizing the aircraft's position to assisting the taxiing and landing process. One of the vital functions of an aircraft autopilot system is stabilization, which keeps the aircraft from stalling or rolling midair. In this paper, we delve deeper regarding how complex eigenvalues play a crucial role in the stabilization function of an autopilot system.

The stability of an aircraft's dynamic system is usually can be modeled by a set of differential equations. Analyzing the eigenvalues of the system's stability matrix leads to an understanding of its behavior. The oscillatory behavior of the system can be detected through complex eigenvalues. If oscillations are exhibited by the system, they can lead to aircraft instability if certain actions are not taken.

This paper focuses on how complex eigenvalues can be used to evaluate the dynamic stability of an aircraft. By analyzing the stability matrix of the system and its eigenvalues, it can be determined whether the aircraft is likely to remain stable or become unstable with the current control settings. The insights from this analysis can improve autopilot algorithms, ensuring they keep the aircraft stable and prevent unwanted oscillations or instability during cruising. The methods discussed in this paper can also be applied in other fields where stability control plays a crucial role in the system performance, helping to create more robust and reliable control systems.

II. Theoretical Basis

A. Matrix

A matrix is a rectangular array of numbers. A matrix with m × n size means that it has m number of rows and n number of columns in the matrix, with m and n both being natural numbers.

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Fig. 2.1 Definition of A Matrix. (Source: [1])

Operations can be done to matrix, such as addition, subtraction, multiplication by a scalar, and multiplication of matrices.

1. Addition and Subtraction

   Matrix addition involves adding the corresponding elements of two matrices. On the other hand, matrix subtraction involves subtracting the corresponding elements of two matrices. These operations can be done only if the matrices have the same dimensions.

$$\mathbf{A+B} = \begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{22} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

Fig. 2.2 Matrix Addition. (Source: [1])

2. Scalar multiplication

   Matrix multiplication by a scalar involves multiplying each element of the matrix by a scalar value.

$$cA = [ca_{ij}] \quad , i = 1, 2, \ldots, m; \; j = 1, 2, \ldots, n$$

Fig. 2.3 Matrix Multiplication by A Scalar. (Source: [1])

3. Multiplication of matrices

   Matrix multiplication can be carried out only if the number of columns in the first matrix matches the number of rows in the second matrix. The resulting product matrix will have dimensions equal to the number of rows in the first matrix and the number of columns in the second matrix. Each element of $M_{ij,}$ is the dot product of the *i*-th row of the first matrix and the *j*-th column of the second matrix.

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{pmatrix}$$

$$\mathbf{C} = \begin{pmatrix} a_{11}b_{11} + \cdots + a_{1n}b_{n1} & a_{11}b_{12} + \cdots + a_{1n}b_{n2} & \cdots & a_{11}b_{1p} + \cdots + a_{1n}b_{np} \\ a_{21}b_{11} + \cdots + a_{2n}b_{n1} & a_{21}b_{12} + \cdots + a_{2n}b_{n2} & \cdots & a_{21}b_{1p} + \cdots + a_{2n}b_{np} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}b_{11} + \cdots + a_{mn}b_{n1} & a_{m1}b_{12} + \cdots + a_{mn}b_{n2} & \cdots & a_{m1}b_{1p} + \cdots + a_{mn}b_{np} \end{pmatrix}$$

Fig. 2.4 Matrix Multiplication. (Source: [1])

B. Determinants

A determinant is a scalar value that can be calculated from a square matrix. A square matrix, commonly written as an n × n matrix, is a matrix where the number of rows equals to the number of columns. The determinant of matrix A is usually written as det(A) or |A|.

$$det\ (A) = \sum_{j=1}^{n} (-1)^{i+j} a_{ij}\, det\ (A_{ij})$$

Fig. 2.5 Definition of A Determinant. (Source: [2])

One important information that the determinant provides is a matrix's invertibility. If the determinant is zero, the matrix is singular, meaning that it cannot be inverted. Otherwise, the matrix is invertable. The determinant of a matrix can also be interpreted as a scaling factor on how the matrix affects the volume of a geometric object, such as a parallelogram (in 2D) or parallelepiped (in 3D), that is formed by vectors.

C. Eigenvalues

An eigenvalue, typically written as λ, is a scalar value that represent the characteristics of a square matrix. It measures shows how the matrix transforms a vector. If $\lambda > 1$, the vector is stretched, whereas if $0 < \lambda < 1$. If $\lambda = 0$, the vector becomes a zero vector. For a square matrix A and a non-zero vector *x*, known as an eigenvector, the eigenvalue can be seen like in the following figure:

$$A\mathbf{x} = \lambda\mathbf{x}$$

Fig. 2.6 Definition of An Eigenvalue. (Source: [3])

A matrix can have multiple eigenvalues and each of them are associated with a corresponding eigenvector. Eigenvalues are found by setting the determinant of the matrix minus λ times the identity matrix equal to zero.

$$\det(\lambda I - A) = 0$$

Fig. 2.7 Eigenvalue Formula. (Source: [3])

D. Complex Numbers

A complex number is a number that consists of two parts: a real part and an imaginary part. Either the real part or the imaginary part can be zero. The imaginary unit is represented by *i*, with $i = \sqrt{-1}$; therefore, $i^2 = -1$.

$$z = a + bi$$

Fig. 2.8 Definition of A Complex Number. (Source: [4])

An Argand diagram is used represent complex numbers in a two-dimensional coordinate system. The horizontal axis represents the real part and the vertical axis represents the imaginary part of the complex number. The complex number is shown as a vector or point in this diagram.

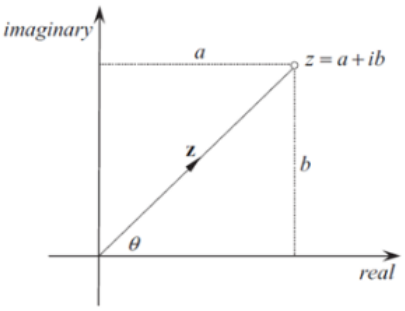

Fig. 2.9 Argand Diagram. (Source: [4])

The distance from the origin to the complex number in the diagram is called a magnitude.

$$|z| = \sqrt{a^2 + b^2}$$

Fig. 2.10 Magnitude Formula. (Source: [4])

The angle θ formed with the real axis is called an argument.

$$\theta = \tan^{-1}(b/a)$$

Fig. 2.11 Argument Formula. (Source: [4])

E. Autopilot Stabilization System

Autopilot is a system in a vehicle that is capable of controlling the vehicle without continuous manual assistance. Nowadays, autopilot is used in many types of vehicles, from cars to aircrafts. Modern autopilot systems are highly advanced, offering various functions depending on the needs of the vehicle.

In an aircraft, autopilot systems can perform many flight maneuvers, such as following the flight plan, assisting in taxiing and landing, and stabilizing the aircraft's heading and altitude. It can minimize human errors and fatigue. Autopilot can also react quicker than humans in situations requiring immediate action, making the flying experience safer.

One of the most important functions of an aircraft autopilot system is stabilization. Stabilization maintains the aircraft's angles of rotation or movements of the airplane to prevent oscillations. An aircraft typically uses a three-axis autopilot system, which balances three rotation angles: roll, pitch, and yaw. Roll refers to the rotation around the longitudinal axis (X-axis) that controls the tilt of the wings. Pitch is the rotation around the lateral axis

(Y-axis) that controls the up-and-down angle of the nose. Yaw is the rotation around the vertical axis (Z-axis) that controls the left-and-right direction of the nose [5]. Stabilization may involve additional axes, such as translation axes, in more complex aircraft dynamics, or fewer axes when less control is needed.

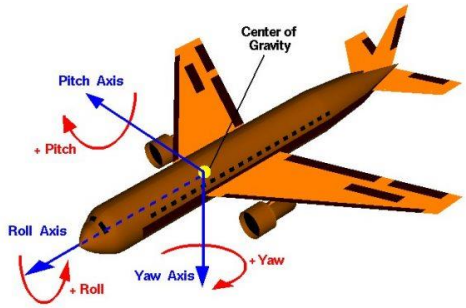

Fig. 2.12 Three-Axis of Rotation. (Source: [5])

Stabilization also deals with oscillation, which is the repetitive back-and-forth movement that can happen in any of the axes. Oscillations are linked to eigenvalues as eigenvalues can be used to determine whether a fixed point is stable or unstable. The stability behavior around the fixed point, in this case, a system, can be determined by the existence of the real part of the eigenvalues, which indicates the amplitude of the oscillations. If the real part is negative, the system is stable as the oscillations will dampen over time. If the real part is positive, the system is unstable since the oscillations will grow, leading to instability. If the real part is zero, the system is marginally stable. It behaves as an undamped oscillator, but this case is not ideal for control systems. On the other hand, the imaginary part of the complex eigenvalues represents the frequency of oscillations, which does not directly affect the stability [6].

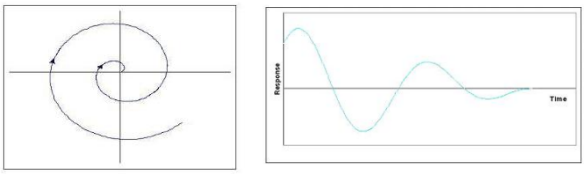

Fig. 2.13 Negative Real Part in Complex Eigenvalues. (Source: [6])

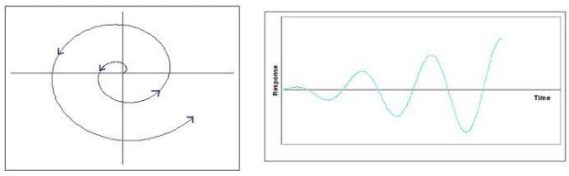

Fig. 2.14 Positive Real Part in Complex Eigenvalues. (Source: [6])

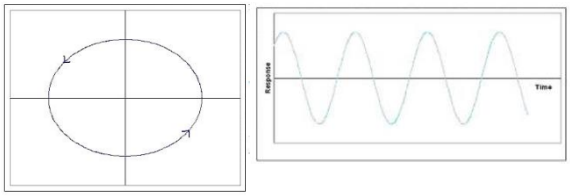

Fig. 2.15 Zero Real Part in Complex Eigenvalues. (Source: [6])

III. METHODOLOGY

Python programming language in utilized in implementing the source code for this problem due to its extensive libraries. Specifically, Python's NumPy library is used in handling matrix operations and calculating eigenvalues. These calculations are performed using a library to ensure high precision, since manual calculations can be complex and error-prone.

The matrix used in this implementation is a square matrix (n × n) with n depending on user's input. The number of rows in matrix A corresponds to the number of control variables in the autopilot system. For example, the user uses the three-axis autopilot system, then the number of control variables is three. Matrix A represents the influence between the control variable, and the value can vary between aircrafts. Each element ranges between -1 and 1 (inclusive), similar to percentages. A negative value indicates a negative correlation, whereas a positive value shows a direct influence.

The following source code allows for input of matrix A:

```
def input_matrix():
  dim = int(input("Amount of control variables:
            "))
  while dim < 1:
   print("The amount of control variables must
     be at least 1. Please try again.")
   dim    =    int(input("Amount    of    control
            variables: "))

  print(f"Input the influence (-1 to 1) between
            control variables in {dim} x {dim}
            matrix form.")
  mat = []
   for i in range(dim):
      while True:
        row = list(map(float, input(f"Row {i+1}:
              ").split()))

        if len(row) != dim:
          print(f"The row must contain
               exactly {dim} elements.
               Please try again.")
          continue

        if all(0 <= abs(x) <= 1 for x in
          row):
          mat.append(row)
          break

        else:
          print("All elements must be
            between -1 and 1 (inclusive).
            Please try again.")
  return np.array(mat)
```

Ideally, matrix A in an aircraft is to maintain stability by manufacturers. However, external factors such as turbulence, electrical malfunctions, or other disturbances can cause imbalances in the system. The stability of matrix A, which directly affects the control system stability, can be evaluated through its eigenvalues. According to the theoretical basis, if the real part of all eigenvalues is negative, the system is considered stable. If any eigenvalue has a positive real part, the system is unstable. If the real part is zero, the system is marginally stable, which is also considered unstable in practice.

The following source code checks system control stability based on matrix A:

```
def check_stability(A):
  eigval = np.linalg.eigvals(A)
  is_stable = all(np.real(e) < 0 for e in
                  eigval)
  return eigval, is_stable
```

If system control is unstable, adjustments may be needed to stabilize the aircraft depending on the initial state control (x_0). The initial state control is a vector with n elements, with the number of elements depending on the configuration of its matrix A. For example, in a tree-axis autopilot system, the control variables—roll, pitch, and yaw—are in radians.

The following source code allows for the input of initial state vector x_0:

```
def input_initial_state(dim):
  while True:
   print(f"Input the initial control state
     vector for each variable ({dim} elements):
     ")
   x = list(map(float, input().split()))
   if len(x) != dim:
     print(f"The vector must contain exactly
      {dim} elements. Please try again.")
     continue
   return np.array(x)
```

The system is considered stable if all elements of initial state vector x_0 are less than 0.1, which is the chosen threshold for oscillation stability. Oscillations under 0.1 are considered to be sufficiently damped over time, as shown in Figure 2.13. Although there is no specific threshold for aircraft stability, it is commonly understood that an aircraft is designed to handle small deviations near zero without significantly affecting its stability, as explained in [7].

If the system is unstable, corrections may be applied to bring the control state closer to stability. The recommended corrections for control are determined by comparing the current state to the desired state, which is a zero vector. Certain limits on the correction are applied in order to prevent extreme changes in aircraft control, though specific limitations vary. As stated in [7], the system should be able to handle moderate changes in control, but extreme adjustments should be avoided, as they can be unpredictable and difficult to manage.

The following source code calculates corrections and applies them:

```
def recommend_correction(status):
  # fixed maximum correction limit
  max_corr_limit = 10
  needed_corr = -status
  # maximum correction limit as 10%
  max_corr = np.maximum(0.1 * np.abs(status),
              max_corr_limit)

  # compares the corrections
  corr = np.sign(needed_corr) *
          np.minimum(np.abs(needed_corr),
          max_corr)
  return corr

def apply_correction(status, corr):
  updated_status = status + corr
  return updated_status
```

The program continuously asks the user whether they wish to apply the recommended corrections while the aircraft remains unstable, this simulates the ongoing stabilization of an aircraft over time. The program stops either when stability is achieved—i.e., when all elements of the state vector x are within the stability threshold—or when the user decides to stop the process.

The following source code shows the overall flow of the program:

```
def main():
  ...
  A = input_matrix()
  eigval, is_stable = check_stability(A)
  print("Eigenvalues:")
  for eig in eigval:
   if np.iscomplex(eig):
    print(f"{eig.real:.3f} + {eig.imag:.3f}j"
     if eig.imag >= 0 else f"{eig.real:.3f} -
     {abs(eig.imag):.3f}j")
   else:
    print(f"{eig.real:.3f} + 0j")

   if is_stable:
    print("System control is stable. No further
     action needed.")
    quit()
   else:
    print("System control is unstable.
     Adjustments may be needed.\n")

  # initial state vector has to match the matrix
    size
  x = input_initial_state(A.shape[0])
  while True:
   print(f"\nCurrent state: {x}")
   # stability threshold
   if not np.all(np.abs(x) < 0.1):
     print("System is unstable.")
     corr = recommend_correction(x)
     print(f"Correction recommendation:
      {corr}")

     # asks about applying the correction
     apply_corr = input("Apply control
                  corrections? (y/n): ")
                  .strip().lower()
     if apply_corr == 'y':
      x = apply_correction(x, corr)
      print(f"State after corrections are
       applied: {x}")
     else:
      print("No corrections applied.")

   else:
     print("System is stable.")
     quit()

   # asks about continuing the simulation
   cont = input("Continue simulation? (y/n):
          ").strip().lower()
   if cont != 'y':
     print("Simulation ended.")
     break
```

IV. Results and Analysis

Testing is conducted on the source code to evaluate its functionality and generate results. These results are then further analyzed to provide insights. The data used in testing are dummy data, consisting of randomly selected

numbers designed to be logical and consistent with related research.

A. Test Case 1: Stable System Control without Complex Parts in Eigenvalues

This test case evaluates a stable system control without complex parts in its eigenvalues. The matrix A used is as follows:

$$A = \begin{bmatrix} -0.6 & 0.3 & 0.2 \\ 0.1 & -0.8 & 0.4 \\ 0.2 & 0.1 & -0.2 \end{bmatrix}$$

The program receives the matrix A input and checks its eigenvalues. Since all of its eigenvalues are negative in the real parts, the program determines that the system control is stable and no further action is needed.

```
------- Dynamic System Stability Simulation -------
Amount of control variables: 3
Input the influence (-1 to 1) between control variables in 3 x 3 matrix form.
Row 1: -0.6 0.3 0.2
Row 2: 0.1 -0.8 0.4
Row 3: 0.2 0.1 -0.2
Eigenvalues:
-0.013 + 0j
-0.787 + 0j
-0.800 + 0j
System control is stable. No further action needed.
```

Fig. 4.1 Test Case 1 Results. (Source: Author)

B. Test Case 2: Stable System Control with Complex Parts in Eigenvalues

This test case evaluates a stable system control with complex parts in its eigenvalues. The matrix A used is as follows:

$$A = \begin{bmatrix} -0.5 & 0.2 & 0.1 \\ 0.1 & -0.7 & 0.3 \\ 0.2 & 0.1 & -0.6 \end{bmatrix}$$

The program receives the matrix A input and checks its eigenvalues. Since all of its eigenvalues are negative in the real parts, the program determines that the system control is stable and no further action is needed. The complex parts do not directly affect the oscillations, so they do not affect the system stability.

```
------- Dynamic System Stability Simulation -------
Amount of control variables: 3
Input the influence (-1 to 1) between control variables in 3 x 3 matrix form.
Row 1: -0.5 0.2 0.1
Row 2: 0.1 -0.7 0.3
Row 3: 0.2 0.1 -0.6
Eigenvalues:
-0.260 + 0j
-0.770 + 0.081j
-0.770 - 0.081j
System control is stable. No further action needed.
```

Fig. 4.2 Test Case 2 Results. (Source: Author)

C. Test Case 3: Unstable System Control and Stable Initial State Vector

This test case evaluates an unstable system control and a stable initial state vector. The matrix A and vector x_0 used are as follows:

$$A = \begin{bmatrix} 0.8 & -0.3 & 0.4 \\ 0.5 & 0.9 & -0.2 \\ -0.4 & 0.3 & 1.0 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} 0.04 \\ -0.05 \\ 0.03 \end{bmatrix}$$

The program receives the matrix A as input and checks its eigenvalues. Since none of its eigenvalues are negative in the real parts, the program determines that the system control is unstable and adjustments may be needed. The program receives the vector x_0 as input and checks its elements. As all elements fall within the stability threshold, indicating that the oscillations are near zero, the system is considered stable.

```
------- Dynamic System Stability Simulation -------
Amount of control variables: 3
Input the influence (-1 to 1) between control variables in 3 x 3 matrix form:
Row 1: 0.8 -0.3 0.4
Row 2: 0.5 0.9 -0.2
Row 3: -0.4 0.3 1.0
Eigenvalues:
0.840 + 0.609j
0.840 - 0.609j
1.020 + 0j
System control is unstable. Adjustments may be needed.

Input the initial control state vector for each variable (3 elements):
0.04 -0.05 0.03

Current state: [ 0.04 -0.05  0.03]
System is stable.
```

Fig. 4.3 Test Case 3 Results. (Source: Author)

D. Test Case 4: Unstable Three-Axis Autopilot System Control and Unstable Initial State Vector

This test case evaluates an unstable three-axis autopilot system control and an unstable initial state vector. The matrix A and vector x_0 used are as follows:

$$A = \begin{bmatrix} 0.8 & -0.3 & 0.4 \\ 0.5 & 0.9 & -0.2 \\ -0.4 & 0.3 & 1.0 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} 12.8 \\ -1.742 \\ 0.06 \end{bmatrix}$$

The program receives the matrix A as input and checks its eigenvalues. Since none of its eigenvalues are negative in the real parts, the program determines that the system control is unstable and adjustments may be needed. The program receives the vector x_0 as input and checks its elements. As not all elements fall within the stability threshold, the system is considered unstable.

```
------- Dynamic System Stability Simulation -------
Amount of control variables: 3
Input the influence (-1 to 1) between control variables in 3 x 3 matrix form:
Row 1: 0.8 -0.3 0.4
Row 2: 0.5 0.9 -0.2
Row 3: -0.4 0.3 1.0
Eigenvalues:
0.840 + 0.609j
0.840 - 0.609j
1.020 + 0j
System control is unstable. Adjustments may be needed.

Input the initial control state vector for each variable (3 elements):
12.8 -1.742 0.06

Current state: [12.8   -1.742  0.06 ]
System is unstable.
```

Fig. 4.4 Test Case 4 Results Part 1. (Source: Author)

Since the system is initially unstable, it recommends specific control corrections based on the current state x_0. When the recommendation is rejected, no corrections are applied.

```
Current state: [12.8   -1.742  0.06 ]
System is unstable.
Correction recommendation: [-10.      1.742  -0.06 ]
Apply control corrections? (y/n): n
No corrections applied.
Continue simulation? (y/n): y
```

Fig. 4.5 Test Case 4 Results Part 2. (Source: Author)

Since no corrections are applied, the current state x remains the same as x_0. With the system still unstable, it

continues to recommend control corrections based on the current state x in order to reach desired state. When the recommendation is accepted, the corrections are applied to the state x.

```
Current state: [12.8   -1.742  0.06 ]
System is unstable.
Correction recommendation: [-10.      1.742  -0.06 ]
Apply control corrections? (y/n): y
State after corrections are applied: [2.8 0.  0. ]
Continue simulation? (y/n): y
```

Fig. 4.6 Test Case 4 Results Part 3. (Source: Author)

The system continuously monitors the state of the system and recommends control corrections based on the updated state x. These corrections are recalculated after every state update. This iterative process continues until the system achieves stability, where the oscillations fall within the set threshold. This process continues as long as the simulation is running and is not interrupted by the user.

```
Current state: [2.8 0.  0. ]
System is unstable.
Correction recommendation: [-2.8  0.   0. ]
Apply control corrections? (y/n): y
State after corrections are applied: [0. 0. 0.]
Continue simulation? (y/n): y

Current state: [0. 0. 0.]
System is stable.
```

Fig. 4.7 Test Case 4 Results Part 4. (Source: Author)

E. Test Case 5: Unstable System Control with Non-Three-Axis Configuration and Unstable Initial State Vector

This test case evaluates an unstable system control with non-three-axis configuration and an unstable initial state vector. The matrix A and vector x_0 used are as follows:

$$A = \begin{bmatrix} 0.5 & 0.7 & -0.8 & 0.6 \\ -0.3 & 1.0 & 0.5 & -0.4 \\ 0.4 & -0.6 & 0.9 & 0.8 \\ -0.5 & 0.3 & -0.7 & 1.0 \end{bmatrix}$$

$$x_0 = \begin{bmatrix} 0.9 \\ -0.8 \\ 0.6 \\ 0.7 \end{bmatrix}$$

The program receives the matrix A with dimensions other than 3 × 3 as input and checks its eigenvalues. Since none of its eigenvalues are negative in the real parts, the program determines that the system control is unstable and adjustments may be needed. The program receives the vector x_0 with a size corresponding to the number of rows in matrix A as input and checks its elements. As none of the elements fall within the stability threshold, the system is considered unstable.

```
------- Dynamic System Stability Simulation -------
Amount of control variables: 4
Input the influence (-1 to 1) between control variables in 4 x 4 matrix form:
Row 1: 0.5 0.7 -0.8 0.6
Row 2: -0.3 1.0 0.5 -0.4
Row 3: 0.4 -0.6 0.9 0.8
Row 4: -0.5 0.3 -0.7 1.0
Eigenvalues:
0.795 + 1.269j
0.795 - 1.269j
0.905 + 0.348j
0.905 - 0.348j
System control is unstable. Adjustments may be needed.
```

Fig. 4.8 Test Case 5 Results Part 1. (Source: Author)

As the system is initially unstable, it recommends certain control corrections based on the current state x_0 that could stabilize the system and reduce oscillations. When the recommendation is rejected, no corrections are applied.

```
Current state: [ 0.9 -0.8  0.6  0.7]
System is unstable.
Correction recommendation: [-0.9  0.8 -0.6 -0.7]
Apply control corrections? (y/n): n
No corrections applied.
Continue simulation? (y/n): y
```

Fig. 4.9 Test Case 5 Results Part 2. (Source: Author)

Since no corrections are applied, the current state x remains the same as x_0. The system continues to recommend control corrections based on the current state x in order to reach desired state when the system is still unstable. However, the process can be interrupted by the user, halting the simulation.

```
Current state: [ 0.9 -0.8  0.6  0.7]
System is unstable.
Correction recommendation: [-0.9  0.8 -0.6 -0.7]
Apply control corrections? (y/n): nn
No corrections applied.
Continue simulation? (y/n): n
Simulation ended.
```

Fig. 4.10 Test Case 5 Results Part 3. (Source: Author)

V. Conclusion

This study successfully analyzes the role of complex eigenvalues in an aircraft dynamic system stability for autopilot control. The developed program replicates the functionality of an autopilot stabilization system and determines system stability by examining the real parts of its matrix eigenvalues. Additionally, it can recommend control adjustments to achieve system stability, where oscillations are near zero, as demonstrated by the test cases conducted.

The program can be further enhanced to handle external factors that affect control like wind, simulating real-life conditions. Using concrete data from experiments in further development could also improve the program's accuracy.

In conclusion, the analysis of aircraft dynamic system stability for autopilot control through complex eigenvalues offers valuable insights regarding the practical application of the complex eigenvalues concept. These insights can contribute to advancements in autopilot algorithms and other stability control fields, helping create more reliable and efficient control systems.

VI. Appendix

The source code for this paper, titled *Analyzing Aircraft Dynamic System Stability for Autopilot Control through Complex Eigenvalues*, can be accessed at:

https://github.com/naomirisaka/Makalah-Algeo

VII. Acknowledgment

First and foremost, the author expresses deep gratitude to God Almighty for His guidance and strength in completing this study. The author would also like to thank Dr. Ir. Rinaldi Munir, M.T., their Linear Algebra and Geometry lecturer, for sharing his knowledge and providing guidance. Lastly, the author expresses heartfelt

thanks to their family and friends for their unwavering encouragement and throughout this study.

REFERENCES

[1] R. Munir, "Aljabar Geometri: Review Matriks," IF2123 Aljabar Linear dan Geometri, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-01-Review-Matriks-2023.pdf. [Accessed: Dec. 23, 2024].
[2] UCL, "Lesson Plan 11," University College London. [Online]. Available: https://www.ucl.ac.uk/~ucahmdl/LessonPlans/Lesson11.pdf. [Accessed: Dec. 23, 2024].
[3] R. Munir, "Aljabar Geometri: Nilai Eigen dan Vektor Eigen (Bagian 1)," IF2123 Aljabar Linear dan Geometri, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf. [Accessed: Dec. 23, 2024].
[4] R. Munir, "Aljabar Geometri: Aljabar Kompleks," IF2123 Aljabar Linear dan Geometri, 2023. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-24-Aljabar-Kompleks-2023.pdf. [Accessed: Dec. 24, 2024].
[5] NASA, "Aircraft Rotations," NASA Glenn Research Center. [Online]. Available: https://www1.grc.nasa.gov/beginners-guide-to-aeronautics/aircraft-rotations/. [Accessed: Dec. 24, 2024].
[6] D. Katzman, J. Moreno, J. Noelanders, and M. Winston-Galant, "Using eigenvalues and eigenvectors to find stability and solve ODEs," *LibreTexts Engineering*. [Online]. Available: https://eng.libretexts.org/Bookshelves/Industrial_and_Systems_Engineering/Chemical_Process_Dynamics_and_Controls_(Woolf)/10%3A_Dynamical_Systems_Analysis/10.04%3A_Using_eigenvalues_and_eigenvectors_to_find_stability_and_solve_ODEs. [Accessed: Dec. 24, 2024].
[7] M. Carley, *Aircraft Stability and Control*, University of Bath, 2020. [Online]. Available: https://people.bath.ac.uk/ensmjc/Notes/stability.pdf. [Accessed: Dec. 24, 2024].

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 1 Januari 2025

Naomi Risaka Sitorus – 13523122