Solving Ordinary Differential Equations using Eigenvalues and Eigenvectors: A Linear Algebra Approach

Daniel Pedrosa Wu - 13523099[1,2]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13523099@std.stei.itb.ac.id, [2]danielpedrosawu5705@gmail.com

***Abstract*—This paper explores the application of linear algebraic principles, specifically eigenvalues, eigenvectors and diagonalization in solving homogeneous linear Ordinary Differential Equations (ODEs). It demonstrates how a high-order ODE can be reduced into a simpler system of equations. Furthermore, the usage of matrix diagonalization will be applied to further simplify the produced matrix into a more manageable form, that is easier to compute. Result shows that these methods are able to effectively compute the solution to various differential equations. While linear algebra is a powerful tool in reducing the complexity of high-order differential equations, it is important to note that the approach does not universally apply to all types of differential equations. This paper emphasizes that linear algebra is just one of many tools in the mathematical toolkit. Regardless, it is a valuable tool, highlighting the importance of a collaborative approach where multiple different disciplines complement each other for efficient problem-solving.**

***Keywords*—eigenvalues, eigenvectors, diagonalization, Ordinary Differential Equations (ODE).**

I. Introduction

Ordinary Differential Equations (ODEs) are important part of our real world. They help model systems across both scientific and non scientific fields. Due to their roots in calculus, solutions are traditionally found using a calculus-based approach. Methods such as variation of parameters or separation of variables are just one of many in solving ODEs using a calculus-based approach. While these techniques are widely used for simpler, low-order ODEs, they have proven to be inconvenient for more complex, high-order ODEs. As the complexities of the problems grow, these calculus-based method has been shown to be more intensive and error-prone.

To solve these problems efficiently, a different approach must be taken. Despite being a different field of mathematics altogether, *linear algebra* has strong ties with calculus. High-order ODEs can be transformed into a matrix-based system of coupled first-order ODEs. This linear algebraic approach allows the usage of techniques such as eigenvalues and eigenvectors decomposition to simplify the problem and find the solution. By leveraging the properties of linear algebra, more efficient solutions can be found, while also being more computationally feasible and less error-prone.

This research explores the application of linear algebra concepts such as eigenvalues, eigenvectors and matrix diagonalization in solving higher-order ODEs. This research aims to demonstrate the strong connection between linear algebra and calculus by taking a linear algebraic approach to an inherently calculus problem. This paper focuses on the theoretical foundation on the concepts used, the mathematics involved and how it can be implemented using the programming language Python.

II. Theoretical Foundation

A. Ordinary Differential Equations

Ordinary Differential Equations are mathematical equations involving a function and its derivatives with respect to a single independent variable. The general form of an ODE is:

$$F\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}\right) = 0 \quad (1)$$

where F is the function, x is the independent variable and y is the dependent variable, with $\frac{dy}{dx}$ as the first derivative of y with respect to x, $\frac{d^2y}{dx^2}$ as the second derivative of y with respect to x and so on.

Order is a common term when talking about ODEs. The order of an ODE is determined by the highest derivative of the dependent variable y with respect to the independent variable x. Higher order ODEs typically describe more intricate behaviors.

In (1), the right side of the equation equals zero. This indicates that the equation is homogeneous. If all terms include the dependent variable y or its derivatives, then the ODE is *homogeneous*. An ODE is *non-homogeneous* if there is a term that is independent of the dependent variable y and its derivatives. Below is an example of a non-homogeneous ODE:

$$F\left(x, y, \frac{dy}{dx}, \frac{d^2y}{dx^2}, \ldots, \frac{d^ny}{dx^n}\right) = g(x) \quad (2)$$

where F is the function, x is the independent variable, y is the dependent variable and $g(x)$ is the non-homogeneous term.

B. First-Order Linear Differential Equations

As explained before, a first-order differential equation involves the first derivative of an unknown function. Consider this first-order linear differential function:

$$\frac{dy}{dx} = ky \quad (3)$$

To find the general solution, one must separate the variables so that the equation becomes:

$$\frac{1}{y}dy = k\,dx$$

Integrate both sides:

$$\int \frac{1}{y}dy = \int k\,dx$$

$$\ln|y| = kx + C$$

To solve for y, exponentiate both sides:

$$e^{\ln|y|} = e^{kx+C}$$

$$|y| = e^{kx+C}$$

$$|y| = e^{kx}e^{C}$$

Let e^C be a new constant C_1:

$$|y| = C_1e^{kx}$$

$$y(x) = C_1e^{kx} \quad (4)$$

This is the general solution for this first-order linear differential equation. If given an initial condition such as $y(0)$, then the solution becomes:

$$y(0) = C_1e^{k\cdot 0}$$

$$y(0) = C_1$$

Substitute C_1 into (4):

$$y(x) = e^{kx}y(0) \quad (5)$$

Therefore, the unique solution to a first-order linear differential equation is obtained.

C. Conversion to a First-Order System

Higher-order ODEs can be converted into a couple system of first-order ODEs. To convert a higher-order ODE into a system of first-order ODEs, new variables that represent the derivative of the dependent variable must be introduced. Consider the following general form of an nth-order ODE:

$$\frac{d^ny}{dx^n} + a_{n-1}\frac{d^{n-1}y}{dx^{n-1}} + \cdots + a_1\frac{dy}{dx} + a_0y = 0 \quad (6)$$

To reduce the order of the equation, new variables y_1, y_2, …, y_n are introduced. Each of these variables represent a derivative of y. The substitutions are as follows:

$$y_1 = y$$

$$y_2 = \frac{dy}{dx}$$

$$\vdots$$

$$y_n = \frac{d^{n-1}y}{dx^{n-1}}$$

Consider the following:

$$\frac{dy_1}{dx} = y_2$$

$$\frac{dy_2}{dx} = y_3$$

$$\vdots$$

$$\frac{dy_{n\text{-}1}}{dx} = y_n$$

$$\frac{dy_n}{dx} = \frac{d^ny}{dx^n}$$

By rearranging (6) and isolating $\frac{d^ny}{dx^n}$ on the left-hand side while moving the other terms to the right-hand side, one gets the following equation:

$$\frac{d^ny}{dx^n} = -a_0y - a_1\frac{dy}{dx} - \cdots - a_{n-1}\frac{d^{n-1}y}{dx^{n-1}} \quad (7)$$

Substituting y_1, y_2, …, y_n into (7) gets the following:

$$\frac{dy_n}{dx} = \frac{d^ny}{dx^n} = -a_0y_1 - a_1y_2 - \cdots - a_{n-1}y_n \quad (8)$$

This system of first-order ODEs can be represented in matrix form as:

$$\frac{d}{dx}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & 1 \\ -a_0 & -a_1 & -a_2 & \cdots & -a_{n-2} & -a_{n-1} \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n-1} \\ y_n \end{bmatrix}$$

$$\frac{d}{dx}\mathbf{y} = \mathbf{A}\mathbf{y} \quad (9)$$

where **y** is a vector of variables and **A** is the coefficient matrix. This approach offers a way for a more efficient solution to solving ODEs.

D. Eigenvalues and Eigenvectors

Eigenvalues and eigenvectors are fundamental concepts in the field of linear algebra. They are crucial in the understanding of linear transformation represented by matrices. Given a square matrix **A** of size $n \times n$, the eigenvectors **x** and the eigenvalues λ of **A** satisfy the following equation:

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x} \quad (10)$$

From this equation, it is observed that the matrix-vector multiplication between the matrix **A** and the vector **x** is equivalent to scaling the vector **x** by a factor of λ. In other words, the eigenvalue is a scalar indicating how much **x** is compressed or stretched while the eigenvector is a nonzero vector that under the linear transformation defined by **A** is scaled by the eigenvalue. Below is an illustration:

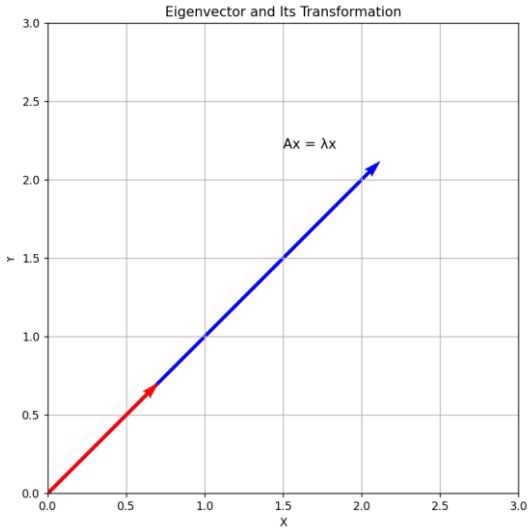


Figure 2.1. Eigenvector and Its Transformation
Source: Author

The eigenvalue represents the magnitude of the transformation, while the eigenvector represents the direction of the transformation. There are 3 main types of transformation which are associated with eigenvalues and eigenvectors which are:

1. If $\lambda > 1$, then the eigenvector is stretched.
2. If $0 < \lambda < 1$, then the eigenvector is compressed.
3. If $\lambda < 0$, then the eigenvector is flipped.

The eigenvalues λ can be found using the characteristic equation below:

$$det(\lambda I - A) = 0 \tag{11}$$

This equation is derived from:

$$\begin{aligned} \mathbf{Ax} &= \lambda \mathbf{x} \\ \mathrm{I}\mathbf{Ax} &= \lambda \mathrm{I}\mathbf{x} \\ \mathbf{Ax} &= \lambda \mathrm{I}\mathbf{x} \\ (\lambda \mathrm{I} - \mathbf{A})\mathbf{x} &= 0 \end{aligned} \tag{12}$$

To ensure a non-trivial solution, the determinant of $(\lambda \mathrm{I} - \mathbf{A})$ must be zero. Thus, the characteristic equation (11) is obtained and the eigenvalues of matrix **A** correspond to the roots of the equation. To find the eigenvectors, one simply substitute the eigenvalue obtained from (11) into (12).

E. Diagonalization

A diagonal matrix is a square matrix in which every element outside of the main diagonal is zero. A diagonal matrix D with size $n \times n$ is written as follows:

$$\mathbf{D} = \begin{bmatrix} d_1 & 0 & 0 & \cdots & 0 \\ 0 & d_2 & 0 & \cdots & 0 \\ 0 & 0 & d_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & d_n \end{bmatrix} \tag{13}$$

A square matrix can be converted into a diagonal matrix through a process called *diagonalization*. Matrix **A** is diagonizable if there exist an eigenvector matrix **P** such that $\mathbf{P^{-1}AP}$ is a diagonal matrix. The formula is as follows:

$$\mathbf{P^{-1}AP = D} \tag{14}$$

An eigenvector matrix is a matrix in which the columns are the eigenvector of a given matrix, in this case matrix **A**. Note that not all square matrices can be converted into a diagonal matrix. For a matrix to be diagonizable, it must have a full set of linearly independent eigenvectors. In the case of a 3×3 matrix, it must have 3 linearly independent eigenvectors to be diagonizable.

F. Matrix Exponentiation

Diagonalizing a matrix simplifies many matrix operations and make subsequent computations much more efficient. One such use is to compute matrix powers. Observe the formula below:

$$\mathbf{A = PDP^{-1}} \tag{15}$$

To find $\mathbf{A}^k$ where k is any positive:

$$\begin{aligned} \mathbf{A}^k &= (\mathbf{PDP^{-1}})^k \\ \mathbf{A}^k &= \mathbf{PD}^k\mathbf{P^{-1}} \end{aligned} \tag{16}$$

The **P** and $\mathbf{P^{-1}}$ will cancel each other out, leaving only one set of **P** and $\mathbf{P^{-1}}$. This is significantly easier to do than directly raising **A** to the power of k. Since D is a diagonal matrix, the process can be done by simply raising the diagonal element to the power of k which is much more efficient to do. (17)

For this research however, diagonalization will be used to simplify ODEs. By applying diagonalization to matrix **A** in (9), the system will be decoupled into independent equations which are much easier to solve. For this purpose, one must understand what an matrix exponential is.

The Taylor series expansion of e^x around $x = 0$ is defined by:

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Another form of this expression is:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \cdots \tag{18}$$

In the context of solving systems of linear differential equations, the matrix exponential $e^{\mathbf{A}}$ is used. For the matrix exponential $e^{\mathbf{A}}$, the same principle as e^x applies. The illustration are as follows:

$$e^{\mathbf{A}} = I + \mathbf{A} + \frac{\mathbf{A}^2}{2!} + \frac{\mathbf{A}^3}{3!} + \frac{\mathbf{A}^4}{4!} + \frac{\mathbf{A}^5}{5!} + \cdots \tag{19}$$

This concept is important because as discussed in Section II.B, the solution for first-order linear differential equation is:

$$y(x) = e^{kx}y(0)$$

where $y(0)$ is the initial condition. Similarly for systems

of linear equation, the solution can be written in terms of matrix exponentials:

$$y(x) = e^{\mathbf{A}x}y(0) \tag{20}$$

where **A** is a coefficient matrix.

Despite that $e^{\mathbf{A}x}$ is still hard to compute, this is where diagonalization can be applied. Using (15), $e^{\mathbf{A}x}$ can be expressed as:

$$e^{\mathbf{A}x} = \mathbf{P}e^{\mathbf{D}x}\mathbf{P^{-1}} \tag{21}$$

Since **D** is a diagonal matrix, $e^{\mathbf{D}x}$ can be expresses as:

$$e^{\mathbf{D}x} = \begin{bmatrix} e^{d_1x} & 0 & 0 & \cdots & 0 \\ 0 & e^{d_2x} & 0 & \cdots & 0 \\ 0 & 0 & e^{d_3x} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & e^{d_nx} \end{bmatrix}$$

Therefore, whenever diagonalization is possible, it is off the best interest to diagonalize the matrix.

G. Finding The Solution

As obtained in Section II.B, the solution of a first-order differential equation can be found using (5). This will be very useful in finding the solution of a higher-order ODEs, as the ODE is now just a system of first-order differential equations. By adapting the solution from (5), the solution of a high-order ODE can be obtained with (20) or the following:

$$y(x) = e^{\mathbf{A}x}y(0)$$

Diagonalization plays a pivotal role in simplifying $e^{\mathbf{A}x}$. By decomposing the **A** matrix into $\mathbf{A} = \mathbf{PDP^{-1}}$, computing $e^{\mathbf{A}x}$ becomes much easier as computing $e^{\mathbf{D}x}$ is easier and less computationally taxing than computing it directly. By applying (21), the final solution is as follows:

$$y(x) = \mathbf{P}e^{\mathbf{D}x}\mathbf{P^{-1}}y(0) \tag{22}$$

III. IMPLEMENTATION

A. Research Limitation

This paper focuses solely on the linear algebraic approach in solving ODEs. To simplify the process, the scope of the problem is intentionally narrowed to exclude calculus-based approaches. The limitations are as follows:

1. Linear Algebra Approach Only: The solution of ODEs will primarily be approached through linear algebraic methods, particularly using eigenvalues and eigenvectors. Although separation of variables method is briefly mentioned, it is solely used as the standard solution for first-order linear differential equation in the form of $\frac{dy}{dx} = ky$ which frequently arises in the linear algebraic approach to solving ODEs.
2. Exclusion of Non-Linear ODEs: Non-linear ODEs often requires different methods to solve. Although non-linear ODEs can be linearized, this process lies beyond the scope of this research so only linear ODEs are considered.
3. Matrix Diagonalization: This paper assumes that the system of ODEs can be diagonalized when converted to matrix notation. Non-diagonalizable matrix are not considered.
4. Homogeneous Systems Only: This paper will only focus on homogeneous ODE systems. Although a linear algebraic approach can be used to find the homogeneous solution in a non-homogeneous system, finding the particular solution typically requires a calculus-based approach.

B. Programming Language

The implementation of this program to solve Ordinary Differential Equations using linear algebra techniques will be carried out by Python. Python is chosen for the simplicity of its syntax and the availability of numerous libraries with powerful numerical tools, such as NumPy, SymPy and SciPy. For this implementation, only NumPy will be used as it provides all the necessary functionality needed to compute the solutions of the ODEs within the scope of this research. Additionally, the *sys* and *json* libraries will be used to help in handling user's input.

```
import numpy as np
import json
import sys
```

C. Input Processing

As mentioned in Section III.B, this implementation accommodates two methods of user interaction: JSON file inputs and terminal-based inputs. This allows user to choose their preferred method in providing the necessary inputs for the program to function. This program receives 3 different input parameters: *coefficient*, *initial conditions* and *the evaluation point* (x).

1. The coefficients represent the coefficient of the Ordinary Differential Equations. The coefficients are provided in descending order of the derivative terms.
2. The initial conditions represent the value of the function and its derivatives at the initial point $x = 0$. The number of initial conditions provided must be equal to the order of the ODE.
3. The evaluation point (x) represent the point where the solution to the ODE will be evaluated.

If a JSON file containing the necessary parameters is provided, the program will call `load_data_from_json` function. This will parse the file and extract the necessary parameters required for the program's execution.

```
def load_data_from_json(filename):
  """Load input from a JSON file"""
  try:
      with open(filename, 'r') as file:
          data = json.load(file)
```

```
      return data
  except FileNotFoundError:
      print("Warning: JSON file not
found. Please provide input manually.")
      return None
```

If a JSON file is not provided or the input is insufficient, the program falls back to a terminal-based input. This is done through the `get_user_input` function which will prompt the user and returns an error to the user if the inputs are not sufficient.

```
def get_user_input():
  """Get user input from the terminal if
a JSON file is not provided"""
  coefficients = list(map(float,
input("Enter the coefficients (space-
separated): ").split()))

  if len(coefficients) < 2:
      print("Error: The differential
equation must atleast be a second-order
differential equation")
      sys.exit(1)

  y0 = list(map(float, input(f"Enter
initial conditions (length should be
{len(coefficients) - 1}): ").split()))

  if len(y0) != len(coefficients) - 1:
      print(f"Error: The length of
initial conditions must be
{len(coefficients) - 1}.")
      sys.exit(1)

  x = float(input("Enter the value of x:
"))

  return coefficients, y0, x
```

Input processing is further handled by the main function where the program ensures the necessary parameters are correctly retrieved.

```
def main():
  if len(sys.argv) == 2:
      """Check if a JSON file is
provided"""
      json_file = sys.argv[1]

      data =
load_data_from_json(json_file)

      if data is None:
          coefficients, y0, x =
get_user_input()
      else:
          coefficients =
data.get('coefficients')
          y0 = data.get('y0')
          x = data.get('x')

          if coefficients is None or
len(coefficients) < 2:
              print("Error: Coefficients
must be provided.")
              sys.exit(1)

          if y0 is None or len(y0) !=
len(coefficients) - 1:
              print(f"Error: Initial
conditions must be provided.")
              sys.exit(1)

          if x is None:
              print("Error: x value must
be provided.")
              sys.exit(1)

  else:
      coefficients, y0, x =
get_user_input()
```

D. Convert to Matrix

To apply linear algebra principles, the given Ordinary Differential Equation must be transformed into a matrix representation as outlined in Section II.C. To achieve this, start by constructing a zero matrix and populating the superdiagonal with ones. This will leave the last row unchanged. The last row is then populated by the negative values of the coefficients in ascending order of the derivatives. This is done by `convert_to_matrix` the function.

```
def convert_to_matrix(coefficients):
  """Convert a differential equation
into a system of coupled first-order
differential equations"""
  order = len(coefficients) - 1
  A = np.zeros((order, order))

  for i in range(order - 1):
      A[i, i + 1] = 1

  for i in range(order):
      A[order - 1, i] = -
coefficients[order - i] /
coefficients[0]

  return A
```

E. Computing the Eigenvalues and Eigenvectors

After converting the ODE into a matrix representation, the next step is to calculate the eigenvalues and eigenvectors. Fortunately, the NumPy library has a built-in function `numpy.linalg.eig` that computes the eigenvalues and eigenvectors of matrix and store it inside an array.

```
eigvals, eigvecs = np.linalg.eig(A) # Extract the eigenvalue and eigenvector using NumPy
```

F. Diagonalizing and Matrix Exponentiation

Using the extracted eigenvalues and eigenvectors, the diagonalized matrix can be constructed. Initially, matrix **P** is formed by placing the eigenvectors as its column vectors. It is very important to note that the matrix **P** must be a square matrix otherwise matrix **A** cannot be diagonalized. Although the scope of this research will only consider diagonalizable matrices, a check is still performed to ensure the matrix is diagonalizable.

Another important step is to calculate the inverse of matrix **P**. This is important to compute the final solution. In order for a matrix to be invertible, the determinant must not equal zero. A diagonal matrix **D** is constructed by populating the main diagonal of a zero matrix with the eigenvalues of matrix **A**.

```
if eig_rows == eig_columns: # Diagonalizable

  # Construct the P matrix
  P = eigvecs

  # Construct the diagonal matrix
  D = np.diag(eigvals)

  if np.linalg.det(P) == 0:
      print("Warning: Matrix P is singular and cannot be inverted.")
      sys.exit(1)
  else:
      P_inv = np.linalg.inv(P)
```

As mentioned in Section II.F, the standard solution for a first-order differential equation can be found using (5). By adapting the formula for matrices, the matrix exponential $e^{\mathbf{A}x}$ is obtained. For more efficient computation, it is decomposed into:

$$e^{\mathbf{A}x} = \mathbf{P}e^{\mathbf{D}x}\mathbf{P}^{-1}$$

As the diagonal matrix **D** has already been obtained, the matrix $e^{\mathbf{D}x}$ can be easily computed.

```
eDx = np.diag(np.exp(D.diagonal() * x))
```

G. Assembling the Solution

To find the solution, all of the necessary components must be inputted to (22):

$$y(x) = \mathbf{P}e^{\mathbf{D}x}\mathbf{P}^{-1}y(0)$$

To ensure that the results are presented in a more manageable form, $y(x)$ is rounded using NumPy's built-in `numpy.round` function.

```
y_x = P @ eDx @ P_inv @ y0

y_x_rounded = np.round(y_x, 4)
```

IV. Results and Analysis

To test the functionality of this program, two equations has been chosen for evaluation, one is a second-order equation while the other is a fourth-order equation.

A. Second-Order Differential Equation

The equation is as follows:

$$y'' - 3y' + 2y = 0$$

with $y(0) = 1$ and $y'(0) = 0$.

The function is going evaluated initially at $x = 0$ to observe the result.

```
Coefficients:
1y'' -3y' +2y = 0

Matrix A (First-Order Representation):
[[ 0.  1.]
 [-2.  3.]]

Matrix P (Eigenmatrix):
[[-0.70710678 -0.4472136 ]
 [-0.70710678 -0.89442719]]

Matrix P (inverted):
[[-2.82842712  1.41421356]
 [ 2.23606798 -2.23606798]]

Diagonalized Matrix:
[[1. 0.]
 [0. 2.]]

Initial conditions:
y1(0) = 1
y2(0) = 0

Solution at x = 0
[ 1. -0.]
```

The vector returned will correspond to y and its derivative in ascending order. As observed, the function correctly returned the value of y and its derivatives when evaluated at $x = 0$. This is consistent with the initial condition $y(0) = 1$ and $y'(0) = 0$. Changing the evaluation point will yield different result which can be done by changing the x value. Below is an example of the

function evaluated at $x = 2$.

```
Coefficients:
1y'' -3y' +2y = 0

Matrix A (First-Order Representation):
[[ 0.  1.]
 [-2.  3.]]

Matrix P (Eigenmatrix):
[[-0.70710678 -0.4472136 ]
 [-0.70710678 -0.89442719]]

Matrix P (inverted):
[[-2.82842712  1.41421356]
 [ 2.23606798 -2.23606798]]

Diagonalized Matrix:
[[1. 0.]
 [0. 2.]]

Initial conditions:
y1(0) = 1
y2(0) = 0

Solution at x = 2
[-39.82   -94.4182]
```

B. Fourth-Order Differential Equation

The equation is as follows:

$$y^{(4)} - 6y''' + 11y'' - 6y' = 0$$

with $y(0) = 2$, $y'(0) = 1$, $y''(0) = 0$ and $y'''(0) = -1$. Similarly to the first equation, it will be initially evaluated at $x = 0$.

```
Matrix A (First-Order Representation):
[[  0.   1.   0.   0.]
 [  0.   0.   1.   0.]
 [  0.   0.   0.   1.]
 [  0.   6. -11.   6.]]

Matrix P (Eigenmatrix):
[[ 1.          0.5         0.10846523 -
0.03492151]
 [ 0.          0.5         0.21693046 -
0.10476454]
 [ 0.          0.5         0.43386092 -
0.31429363]
 [ 0.          0.5         0.86772183 -
0.9428809 ]]

Matrix P (inverted):
[[  1.          -1.83333333   1.
-0.16666667]
 [  0.           6.          -5.
1.        ]
 [ -0.         -13.82931669  18.43908891
-4.60977223]
 [  0.          -9.54521404  14.31782106
-4.77260702]]

Diagonalized Matrix:
[[0. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 2. 0.]
 [0. 0. 0. 3.]]

Initial conditions:
y1(0) = 2
y2(0) = 1
y3(0) = 0
y4(0) = -1

Solution at x = 0
[ 2.  1. -0. -1.]
```

Just like in the first equation, this equation correctly returns the initial condition when evaluated at $x = 0$. Now, try evaluating at $x = 3$:

```
  Coefficients:
  1y^(4) -6y''' +11y'' -6y' = 0

  Matrix A (First-Order Representation):
  [[  0.   1.   0.   0.]
   [  0.   0.   1.   0.]
   [  0.   0.   0.   1.]
   [  0.   6. -11.   6.]]

  Matrix P (Eigenmatrix):
  [[ 1.          0.5         0.10846523
-0.03492151]
   [ 0.          0.5         0.21693046
-0.10476454]
   [ 0.          0.5         0.43386092
-0.31429363]
   [ 0.          0.5         0.86772183
-0.9428809 ]]

  Matrix P (inverted):
  [[  1.          -1.83333333   1.
-0.16666667]
   [  0.           6.          -5.
1.        ]
   [ -0.         -13.82931669
18.43908891  -4.60977223]
   [  0.          -9.54521404
14.31782106  -4.77260702]]

  Diagonalized Matrix:
  [[0. 0. 0. 0.]
   [0. 1. 0. 0.]
   [0. 0. 2. 0.]
   [0. 0. 0. 3.]]
```

```
  Initial conditions:
  y1(0) = 2
  y2(0) = 1
  y3(0) = 0
  y4(0) = -1

  Solution at x = 3
[  997.6324  3294.8982 10591.1246
33286.6612]
```

C. Potential Improvements

As stated in Section III.A, this method is not without its fault. This technique can only be used to solve linear homogeneous ODE. In order to solve non-linear or non-homogeneous ODE, one must employ other calculus-based techniques to solve the differential equations. Certain matrix also cannot be diagonalized rendering this method unable to solve those problems.

V. Conclusion

Linear algebra has proven to be an incredible powerful tool in solving Ordinary Differential Equations. By leveraging matrix properties, it is able to dissect a high-order equations into simpler system of first-order ones. While it is powerful, its important to know that it is not a one-size-for-all solution. Linear algebraic methods are limited to certain types of ODEs, particularly those that are diagonizable. Many problems that involve ODE in the real world can involve non-linear, non-diagonizable or more complex ODEs. Collaboration with other mathematical tools and methods are important for such scenario. Linear algebra methods are enchanced when combined with other approaches. A collaborative approach must be taken to solve problem effectively and efficiently.

VI. Acknowledgment

The author expresses gratitude to lecturer Ir. Rila Mandaa, M. Eng, Ph. D as the lecturer of IF2123 Linear Algebra and Geometry course, for his guidance in finishing this paper.

VII. Appendix

Full source code of the program is available at https://github.com/DanielDPW/Makalah_IF2123_Algeo

VII. References

[1] R. Munir. “Nilai Eigen dan Vektor Eigen (Bagian 1)”, 2023. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/20 23-2024/Algeo-19-Nilai-Eigen-dan-Vektor-Eigen-Bagian1-2023.pdf (accessed: Dec. 29, 2024).

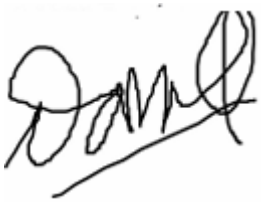
[2] R. Munir. “Nilai Eigen dan Vektor Eigen (Bagian 2)”, 2023. https://informatika.stei.itb.ac.id/~rinaldi.munir/AljabarGeometri/2023-2024/Algeo-20-Nilai-Eigen-dan-Vektor-Eigen-Bagian2-2023.pdf (accessed: Dec. 29, 2024).

[3] M. W. Hirsch, S. Smale, and R. L. Devaney, *Differential Equations, Dynamical Systems, and an Introduction to Chaos*. 2nd ed. Boston, MA: Academic Press, 2004.

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, 27 Desember 2024



Daniel Pedrosa Wu 13523099