PROGRAM STUDI INFORMATIKA
FAKULTAS TEKNIK DAN INFORMATIKA
UNIVERSITAS MULTIMEDIA NUSANTARA
SEMESTER GENAP TAHUN AJARAN 2021/2022

IF420 – ANALISIS NUMERIK

Pertemuan ke 5 – Eigenvalues and Eigenvectors

Seng Hansun, S.Si., M.Cs.

Capaian Pembelajaran Mingguan Mata Kuliah (Sub-CPMK):

Sub-CPMK 5: Mahasiswa mampu memahami dan menerapkan teknik mencari nilai dan vektor Eigen – C3

Reviews

- Basics of Linear Algebra
- Linear Transformations
- Systems of Linear Equations
- Solutions to Systems of Linear Equations
- Solve Systems of Linear Equations in Python
- Matrix Inversion

Outlines

- Eigenvalues and Eigenvectors Problem Statement
- The Power Method
- The QR Method
- Eigenvalues and Eigenvectors in Python

Motivation

- Today, we are going to learn about **eigenvalues** and **eigenvectors** which play a very important role in many applications in science and engineering.
- The prefix **eigen-** is adopted from the German word eigen for "**proper**", "**characteristic**" and it may sound really abstract and scary at beginning.
- But when you start to understand them, you will find that they bring in a lot of insights and conveniences into our problems.
- They have many applications, to name a few, finding the **natural frequencies** and **mode shapes** in dynamics systems, solving **differential equations**, **reducing the dimensions** using **principal components analysis**, getting the **principal stresses** in the mechanics, and so on. Even the famous Google's search engine algorithm - **PageRank**, uses the eigenvalues and eigenvectors to assign scores to the pages and rank them in the search.

The Problem Statement

- We learned from last week that matrix A apply to column vector x, that is Ax, is a **linear transformation** of x.
- There is a **special transform** in the following form:

$$Ax = \lambda x$$

  where A is $n \times n$ matrix, x is $n \times 1$ column vector ($X \neq 0$), and λ is some scalar.
- Any λ that satisfies the above equation is known as an **eigenvalue** of the matrix A, while the associated vector x is called an **eigenvector** corresponding to λ.

The Motivation Behind

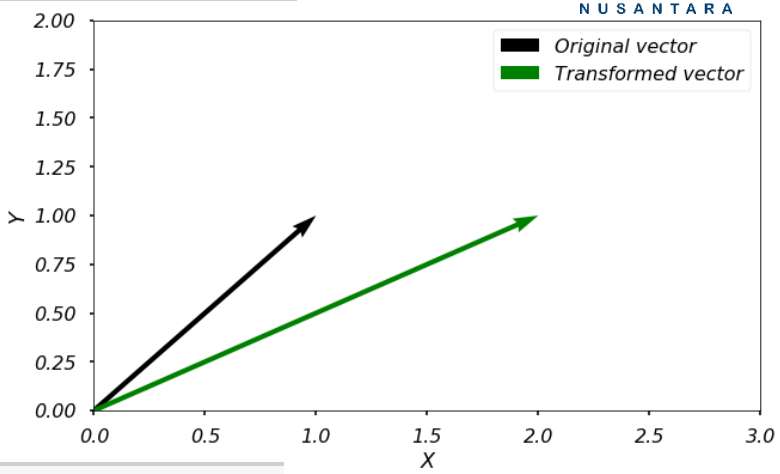
- The motivation behind the eigenvalues and eigenvectors is that, it helps us to understand the **characteristics** **of the** **linear transformation**, thus make things easy.
- We know that a vector x can be **transformed** to a **different vector** by multiplying A - $\boldsymbol{Ax}$.
- The effect of the transformation represents a **scale** of the length of the vector and/or the **rotate** of the vector.
- The above equation points out that for some vectors, the effect of transformation of Ax is only scale (**stretching, compressing**, and **flipping**).
- The **eigenvectors** are the vectors that have this property and the **eigenvalues** λ's are the **scale** **factors**.
- Let's look at the following example.

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

plt.style.use('seaborn-poster')

%matplotlib inline

def plot_vect(x, b, xlim, ylim):
    '''
    function to plot two vectors,
    x - the original vector
    b - the transformed vector
    xlim - the limit for x
    ylim - the limit for y
    '''
    plt.figure(figsize = (10, 6))
    plt.quiver(0,0,x[0],x[1],\
        color='k',angles='xy',\
        scale_units='xy',scale=1,\
        label='Original vector')
    plt.quiver(0,0,b[0],b[1],\
        color='g',angles='xy',\
        scale_units='xy',scale=1,\
        label ='Transformed vector')
    plt.xlim(xlim)
    plt.ylim(ylim)
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.legend()
    plt.show()
```

In [2]:

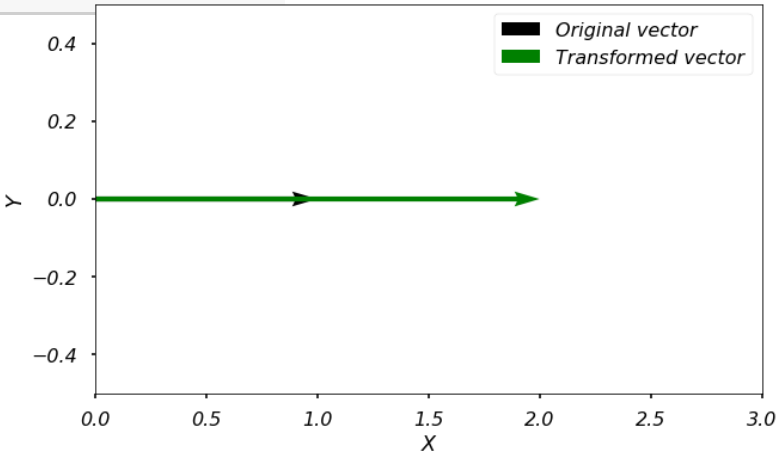
```
A = np.array([[2, 0],[0, 1]])

x = np.array([[1],[1]])
b = np.dot(A, x)
plot_vect(x,b,(0,3),(0,2))
```



- The generated figure show that the original vector x is rotated and stretched longer after transformed by A.

In [3]:

```
x = np.array([[1], [0]])
b = np.dot(A, x)

plot_vect(x,b,(0,3),(-0.5,0.5))
```



- The new vector is [[2], [0]], therefore, the transform is $Ax = 2x$ with $x = [[1], [0]]$ and $\lambda = 2$.

The Characteristic Equation

- In order to get the **eigenvalues** and **eigenvectors**, from $Ax = \lambda x$, we can get the following form:

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

where I is the **identify** matrix with the same dimensions as A.

- If matrix $A - \lambda I$ has an inverse, then multiply both sides with $(A - \lambda I)^{-1}$, we get a **trivial solution** $x = 0$. Therefore, when $A - \lambda I$ is **singular** (no inverse exist), we have a **nontrivial solution**, which means that the **determinant** is **zero**:

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

- This equation is called **characteristic equation**, which will lead to a **polynomial** equation for λ, then we can solve for the **eigenvalues**.
- Let's look at one example.

The Characteristic Equation

- **Example**: Get the eigenvalues for matrix [[0, 2], [2, 3]]
- The characteristic equation gives us

$$\begin{vmatrix} 0-\lambda & 2 \\ 2 & 3-\lambda \end{vmatrix} = 0$$

- Therefore, we have

$$-\lambda(3-\lambda) - 4 = 0 \Rightarrow \lambda^2 - 3\lambda - 4 = 0$$

- We get two eigenvalues:

$$\lambda_1 = 4, \qquad \lambda_2 = -1$$

- **Example**: Get the eigenvectors for the above two eigenvalues.
- Let's get the first eigenvector when $\lambda_1 = 4$, we can simply insert it back to $(A - \lambda I)x = 0$, where we have:

$$\begin{bmatrix} -4 & 2 \\ 2 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- Therefore, we have two equations $-4x_1 + 2x_2 = 0$ and $2x_1 - x_2 = 0$, both of them indicate that $x_2 = 2x_1$. Therefore, we can have the first eigenvector as

$$x = k_1 \begin{bmatrix} 1 \\ 2 \end{bmatrix} \quad \leftarrow \quad \text{First } x$$

- k_1 is a scalar vector ($k_1 \neq 0$), as long as we have the ratio between x_2 and x_1 as 2, it will be an **eigenvector**. We can verify the vector [[1], [2]] is an eigenvector by inserting it back:

$$\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \end{bmatrix} = 4 \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

- By inserting $\lambda_2 = -1$ similarly as above, we can get the other eigenvector as the following, where $k_2 \neq 0 : x = k_2 \begin{bmatrix} -2 \\ 1 \end{bmatrix} \quad \leftarrow \quad$ Second x.

The Power Method

- In some problems, we only need to find the **largest dominant eigenvalue** and its **corresponding eigenvector**. In this case, we can use the **Power method** - a iterative method that will converge to the **largest eigenvalue**.
- Consider an $n \times n$ matrix A that has n **linearly independent** real eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$ and the corresponding eigenvectors $v_1, v_2, \ldots, v_n$. Since the eigenvalues are scalars, we can rank them so that $|\lambda_1| > |\lambda_2| > \cdots > |\lambda_n|$ (actually, we only require $|\lambda_1| > |\lambda_2|$, other eigenvalues may be equal to each other).
- Because the eigenvectors are **independent**, they are a set of basis vectors, which means that any vector that is in the same space can be written as a **linear combination** of the basis vectors. That is, for any vector x_0, it can be written as:

$$x_0 = c_1 v_1 + c_2 v_2 + \cdots + c_n v_n$$

where $c_1 \neq 0$ is the constraint. If it is zero, then we need to choose another initial vector so that $c_1 \neq 0$.

- Now let's multiply both sides by A:

$$Ax_0 = c_1Av_1 + c_2Av_2 + \cdots + c_nAv_n$$

- Since $Av_i = \lambda v_i$, we will have:

$$Ax_0 = c_1\lambda_1 v_1 + c_2\lambda_2 v_2 + \cdots + c_n\lambda_n v_n$$

- We can change the above equation to:

$$Ax_0 = c_1\lambda_1\left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2}{\lambda_1}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n}{\lambda_1}v_n\right] = c_1\lambda_1 x_1$$

where x_1 is a new vector and $x_1 = v_1 + \frac{c_2}{c_1}\frac{\lambda_2}{\lambda_1}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n}{\lambda_1}v_n$.

- This finishes the first iteration. And we can multiply A to x_1 to start the 2nd iteration:

$$Ax_1 = \lambda_1 v_1 + \frac{c_2}{c_1}\frac{\lambda_2^2}{\lambda_1}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^2}{\lambda_1}v_n$$

- Similarly, we can rearrange the above equation to:

$$Ax_1 = \lambda_1\left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2^2}{\lambda_1^2}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^2}{\lambda_1^2}v_n\right] = \lambda_1 x_2$$

where x_2 is another new vector and $x_2 = v_1 + \frac{c_2}{c_1}\frac{\lambda_2^2}{\lambda_1^2}v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^2}{\lambda_1^2}v_n$.

- We can continue multiply A with the new vector we get from each iteration k times:

$$Ax_{k-1} = \lambda_1 \left[v_1 + \frac{c_2}{c_1}\frac{\lambda_2^k}{\lambda_1^k} v_2 + \cdots + \frac{c_n}{c_1}\frac{\lambda_n^k}{\lambda_1^k} v_n \right] = \lambda_1 x_k$$

- Because λ_1 is the **largest eigenvalue**, therefore, the ratio $\frac{\lambda_i}{\lambda_1} < 1$ for all $i > 1$. Thus when we increase k to sufficient large, the ratio of $\left(\frac{\lambda_n}{\lambda_1}\right)^k$ will be close to 0. So that all the terms that contain this ratio can be neglected as k grows:

$$Ax_{k-1} = \lambda_1 v_1$$

- Essentially, as k is large enough, we will get the **largest** eigenvalue and its corresponding eigenvector. When implementing this **Power method**, we usually **normalize** the resulting vector in each iteration. This can be done by factoring out the largest element in the vector, which will make the largest element in the vector equal to 1. This **normalization** will get us the **largest eigenvalue** and its **corresponding eigenvector** at the same time.
- Let's take a look of the following example.

The Power Method

- **Example**: We know from last section that the largest eigenvalue is 4 for matrix $A = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$, now use the Power method to find the largest eigenvalue and the associated eigenvector. You can use the initial vector [1, 1] to start the iteration.
- 1st iteration:

$$\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \end{bmatrix} = 5 \begin{bmatrix} 0.4 \\ 1 \end{bmatrix}$$

- 2nd iteration:

$$\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 0.4 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3.8 \end{bmatrix} = 3.8 \begin{bmatrix} 0.5263 \\ 1 \end{bmatrix}$$

- 3rd iteration:

$$\begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix} \begin{bmatrix} 0.5263 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 4.0526 \end{bmatrix} = 4.0526 \begin{bmatrix} 0.4935 \\ 1 \end{bmatrix}$$

The Power Method

- 4th iteration:

$$\begin{bmatrix}0 & 2\\ 2 & 3\end{bmatrix}\begin{bmatrix}0.4935\\ 1\end{bmatrix} = \begin{bmatrix}2\\ 3.987\end{bmatrix} = 3.987\begin{bmatrix}0.5016\\ 1\end{bmatrix}$$

- 5th iteration:

$$\begin{bmatrix}0 & 2\\ 2 & 3\end{bmatrix}\begin{bmatrix}0.5016\\ 1\end{bmatrix} = \begin{bmatrix}2\\ 4.0032\end{bmatrix} = 4.0032\begin{bmatrix}0.4996\\ 1\end{bmatrix}$$

- 6th iteration:

$$\begin{bmatrix}0 & 2\\ 2 & 3\end{bmatrix}\begin{bmatrix}0.4996\\ 1\end{bmatrix} = \begin{bmatrix}2\\ 3.9992\end{bmatrix} = 3.9992\begin{bmatrix}0.5001\\ 1\end{bmatrix}$$

- 7th iteration:

$$\begin{bmatrix}0 & 2\\ 2 & 3\end{bmatrix}\begin{bmatrix}0.5001\\ 1\end{bmatrix} = \begin{bmatrix}2\\ 4.0002\end{bmatrix} = 4.0002\begin{bmatrix}0.5000\\ 1\end{bmatrix}$$

We can see after 7 iterations, the eigenvalue converged to 4 with [0.5, 1] as the corresponding eigenvector.

The Power Method

- Let's try to implement the **Power method** in Python!

In [4]:
```
import numpy as np

def normalize(x):
    fac = abs(x).max()
    x_n = x / x.max()
    return fac, x_n
```

In [5]:
```
x = np.array([1, 1])
a = np.array([[0, 2],
              [2, 3]])

for i in range(8):
    x = np.dot(a, x)
    lambda_1, x = normalize(x)

print('Eigenvalue:', lambda_1)
print('Eigenvector:', x)
```

```
Eigenvalue: 3.999949137887188
Eigenvector: [0.50000636 1.        ]
```

The Inverse Power Method

- The **eigenvalues** of the **inverse matrix** A^{-1} are the **reciprocals** of the **eigenvalues** of A.
- We can take advantage of this feature as well as the Power method to get the **smallest eigenvalue** of A, this will be basis of the **Inverse Power method**.
- The steps are very simple, instead of multiplying A as described above, we just multiply A^{-1} for our iteration to find the **largest** value of $\frac{1}{\lambda_1}$, which will be the **smallest** value of the eigenvalues for A.
- As for the inverse of the matrix, in practice, we can use the methods we covered in the previous lesson to calculate it. We won't got to the details here, but let's see an example.

The Inverse Power Method

- **Example**: Find the smallest eigenvalue and eigenvector for $A = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$.

In [6]:
```
from numpy.linalg import inv
```

In [7]:
```
x = np.array([1, 1])
a = np.array([[0, 2],
              [2, 3]])

a_inv = inv(a)

for i in range(8):
    x = np.dot(a_inv, x)
    lambda_1, x = normalize(x)

print('Eigenvalue:', lambda_1)
print('Eigenvector:', x)
```

```
Eigenvalue: 1.99931360585723
Eigenvector: [ 1.         -0.49994278]
```

The QR Method

- The **QR method** is a preferred iterative method to find **all the eigenvalues** of a matrix (but **not the eigenvectors** at the same time). The idea is based on the following two concepts:

1. **Similar matrices** will have the **same eigenvalues** and **associated eigenvectors**. Two square matrices A and B are similar if:

$$A = C^{-1}BC$$

   where C is an invertible matrix.

2. The **QR method** is a way to **decompose** a matrix into two matrices Q and R, where Q is an **orthogonal matrix**, and R is an **upper triangular matrix**. An orthogonal matrix has the features: $Q^{-1} = Q^T$, which means $Q^{-1}Q = Q^TQ = I$.

- How do we link these two concepts to find the eigenvalues?

- Say we have a matrix A_0 whose eigenvalues must be determined.
- At the k-th step (starting with $k = 0$), we can perform the **QR decomposition** and get $A_k = Q_k R_k$, where Q_k is an **orthogonal** matrix and R_k is an **upper triangular** matrix.
- We then form $\boldsymbol{A_{k+1} = R_k Q_k}$, which we note that

$$A_{k+1} = R_k Q_k = Q_k^{-1} Q_k R_k Q_k = Q_k^{-1} A_k Q_k$$

- Therefore, all the A_k are **similar**, as we discussed above they all have the **same eigenvalues**.
- As the iteration goes, we will eventually **converge** to an **upper triangular matrix** form:

$$A_k = R_k Q_k = \begin{bmatrix} \lambda_1 & X & \cdots & X \\ 0 & \lambda_2 & \cdots & X \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}$$

  where the **diagonal values** are the **eigenvalues** of the matrix.
- In each iteration of the QR method, factoring a matrix into an **orthogonal** and an **upper triangular** matrix can be done by using a special matrix called **Householder matrix**. We will not go into the mathematical details how to get the Q and R from the matrix, instead, we will use the Python function to obtain the two matrices directly.

The QR Method

- **Example**: Use the **qr** function in **numpy.linalg** to decompose matrix $A = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$ and verify the results.

In [8]:
```
import numpy as np
from numpy.linalg import qr
```

In [9]:
```
a = np.array([[0, 2],
              [2, 3]])

q, r = qr(a)

print('Q:', q)
print('R:', r)

b = np.dot(q, r)
print('QR:', b)
```

```
Q: [[ 0. -1.]
 [-1.  0.]]
R: [[-2. -3.]
 [ 0. -2.]]
QR: [[0. 2.]
 [2. 3.]]
```

The QR Method

- **Example**: Use the QR method to get the eigenvalues of matrix $A = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$. Do 20 iterations, and print out the 1st, 5th, 10th, and 20th iteration.

```
In [10]:
a = np.array([[0, 2],
              [2, 3]])
p = [1, 5, 10, 20]
for i in range(20):
    q, r = qr(a)
    a = np.dot(r, q)
    if i+1 in p:
        print(f'Iteration {i+1}:')
        print(a)
```

```
Iteration 1:
[[3. 2.]
 [2. 0.]]
Iteration 5:
[[ 3.99998093  0.00976559]
 [ 0.00976559 -0.99998093]]
Iteration 10:
[[ 4.00000000e+00  9.53674316e-06]
 [ 9.53674316e-06 -1.00000000e+00]]
Iteration 20:
[[ 4.00000000e+00  9.09484250e-12]
 [ 9.09494702e-12 -1.00000000e+00]]
```

- We can see after the 5th iteration, the eigenvalues are converge to the correct ones.

Eigenvalues and Eigenvectors in Python

- Though the methods we introduced so far look complicated, the actual calculation of the eigenvalues and eigenvectors in Python is fairly easy.
- The main built-in function in Python to solve the eigenvalue/eigenvector problem for a **square array** is the **eig** function in **numpy.linalg**.
- **Example**: Calculate the eigenvalues and eigenvectors for matrix $A = \begin{bmatrix} 0 & 2 \\ 2 & 3 \end{bmatrix}$.

In [11]:
```
import numpy as np
from numpy.linalg import eig
```

In [12]:
```
a = np.array([[0, 2],
              [2, 3]])
w,v=eig(a)
print('E-value:', w)
print('E-vector', v)
```

```
E-value: [-1.  4.]
E-vector [[-0.89442719 -0.4472136 ]
 [ 0.4472136  -0.89442719]]
```

Eigenvalues and Eigenvectors in Python

- **Example**: Compute the eigenvalues and eigenvectors for matrix $A = \begin{bmatrix} 2 & 2 & 4 \\ 1 & 3 & 5 \\ 2 & 3 & 4 \end{bmatrix}$.

```
In [13]: a = np.array([[2, 2, 4],
                       [1, 3, 5],
                       [2, 3, 4]])
         w,v=eig(a)
         print('E-value:', w)
         print('E-vector', v)
```

```
E-value: [ 8.80916362  0.92620912 -0.73537273]
E-vector [[-0.52799324 -0.77557092 -0.36272811]
 [-0.604391    0.62277013 -0.7103262 ]
 [-0.59660259 -0.10318482  0.60321224]]
```

Practice

1. Write down the characteristic equation for matrix $A = \begin{bmatrix} 3 & 2 \\ 5 & 3 \end{bmatrix}$.
2. Using the above characteristic equation to solve for eigenvalues and eigenvectors for matrix A.
3. Use the first eigenvector that derived from problem 2 to verify that $Ax = \lambda x$.

Next Week's Outline

- Least Squares Regression Problem Statement
- Least Squares Regression Derivation (Linear Algebra)
- Least Squares Regression Derivation (Multivariable Calculus)
- Least Squares Regression in Python
- Least Squares Regression for Nonlinear Functions

References

- Kong, Qingkai; Siauw, Timmy, and Bayen, Alexandre. 2020. Python Programming and Numerical Methods: A Guide for Engineers and Scientists. Academic Press. https://www.elsevier.com/books/python-programming-and-numerical-methods/kong/978-0-12-819549-9
- Other online and offline references

Visi

Menjadi Program Studi Strata Satu Informatika **unggulan** yang menghasilkan lulusan **berwawasan internasional** yang **kompeten** di bidang Ilmu Komputer (*Computer Science*), **berjiwa wirausaha** dan **berbudi pekerti luhur**.

INFORMATIKA
UMN
UNIVERSITAS
MULTIMEDIA
NUSANTARA

Misi

1. Menyelenggarakan pembelajaran dengan teknologi dan kurikulum terbaik serta didukung tenaga pengajar profesional.
2. Melaksanakan kegiatan penelitian di bidang Informatika untuk memajukan ilmu dan teknologi Informatika.
3. Melaksanakan kegiatan pengabdian kepada masyarakat berbasis ilmu dan teknologi Informatika dalam rangka mengamalkan ilmu dan teknologi Informatika.