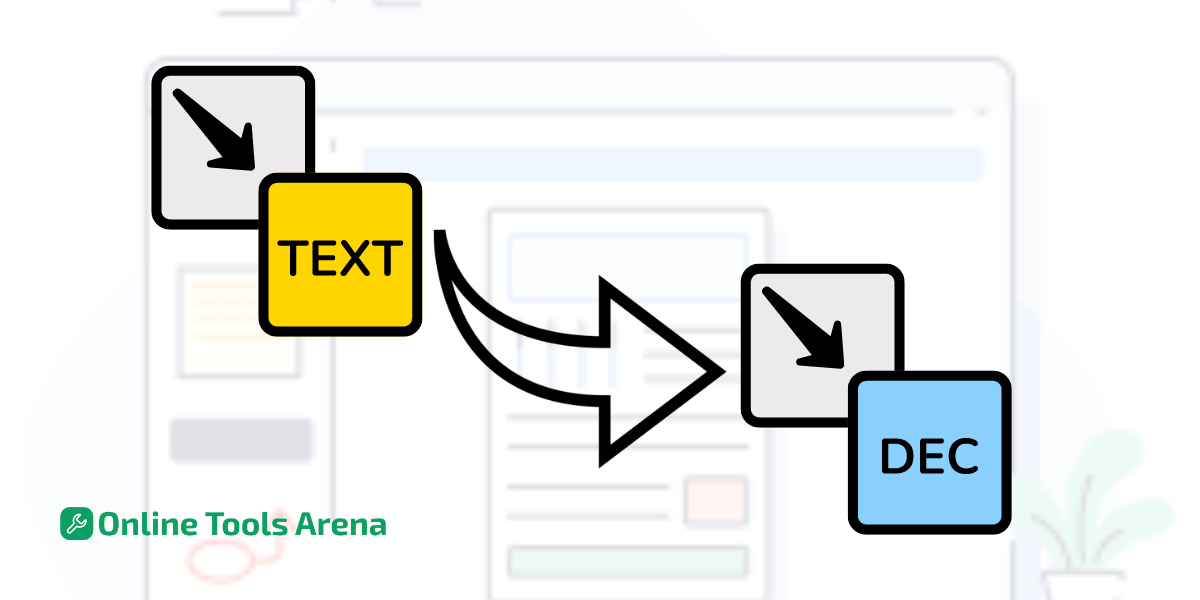
Text Representation Techniques: Bag of Words, TF-IDF

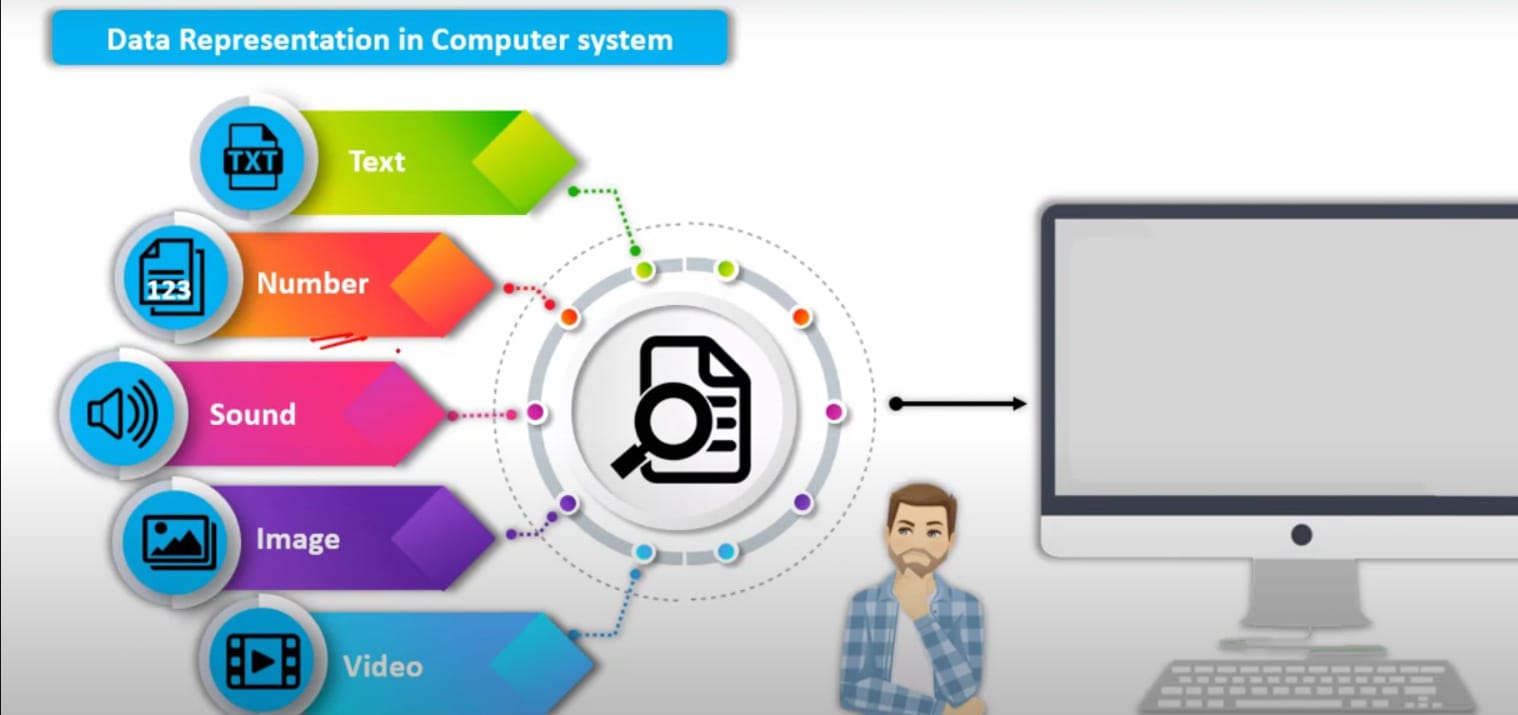
# Data Representation

Data representation is the way information is structured and encoded so that it can be processed by computers or interpreted by humans. It transforms raw data into a meaningful format, like converting text into numerical vectors for machine learning or images into pixel grids.

## Example:



**Types of Data Representation**



# Text Representation :

Understanding how to represent text data in a way that can be used by machine learning algorithms is crucial in natural language processing (NLP). Two fundamental techniques for text representation are Bag of Words (BoW) and TF-IDF (Term Frequency-Inverse Document Frequency).

# 1. Bag of Words (BoW)

## Concept:

Bag of Words is a simple method to represent text data. It involves creating a vocabulary of all the unique words present in the entire text corpus and then representing each document by counting the occurrence of each word from this vocabulary. The order of words is ignored, and only the frequency of words matters.

## Step-by- Step

**Step 1: Collecting the Text Data**

First, gather the text data you want to process. Let’s use two example sentences:

Example:

"The cat sat on the mat."

"The dog sat on the log.”

**Step 2: Tokenization**

Tokenization involves breaking the text into individual words, known as tokens. For our example, the tokenized sentences would be:

Example:

['The', 'cat', 'sat', 'on', 'the', 'mat']

['The', 'dog', 'sat', 'on', 'the', 'log'

**Step 3: Building the Vocabulary**

Next, create a list of all unique words (vocabulary) from the entire text corpus. Thevocabulary for our sentences is:

Example:

['The', 'cat', 'sat', 'on', 'the', 'mat', 'dog', 'log']

**Step 4: Encoding Sentences as Vectors**

Each sentence is represented as a vector, with each position corresponding to a word in the vocabulary. The value at each position is the count of that word in the sentence.

**Example Vectors:**

For the sentence "The cat sat on the mat.":

* The vocabulary words are ['The', 'cat', 'sat', 'on', 'the', 'mat', 'dog', 'log'].
* The vector representation is [2, 1, 1, 1, 1, 1, 0, 0].
  + 'The' appears 2 times
  + 'cat' appears 1 time
  + 'sat' appears 1 time
  + 'on' appears 1 time
  + 'the' appears 1 time
  + 'mat' appears 1 time
  + 'dog' appears 0 times
  + 'log' appears 0 times

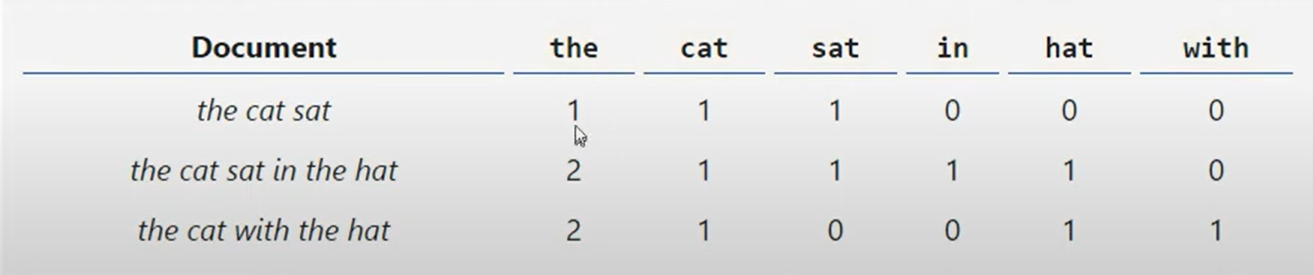
For the sentence "The dog sat on the log.":

* The vocabulary words are ['The', 'cat', 'sat', 'on', 'the', 'mat', 'dog', 'log'].
* The vector representation is [2, 0, 1, 1, 1, 0, 1, 1].
  + 'The' appears 2 times
  + 'cat' appears 0 times
  + 'sat' appears 1 time
  + 'on' appears 1 time
  + 'the' appears 1 time
  + 'mat' appears 0 times
  + 'dog' appears 1 time
  + 'log' appears 1 time

## Example:

Suppose you have the following three sentences:  
1. "the cat sat"  
2. "the cat sat in the hat"

3. "the cat with the hat"  
  
Step-by-Step Process:  
1. Build Vocabulary:  
 - Vocabulary = ["cat", "hat", "in", "sat", "the", "with"]  
2. Create Frequency Vectors:



- Sentence 1: "the cat sat"  
 - [1, 0, 0, 1, 1, 0]  
- Sentence 2: "the cat sat in the hat"  
 - [1, 1, 1, 1, 2, 0]

- Sentence 3: "the cat with the hat"  
 - [1, 1, 0, 0, 2, 1]

## Code :

from sklearn.feature\_extraction.text import CountVectorizer

documents = ["the cat sat", "the cat sat in the hat", "the cat with the hat"]

vectorizer = CountVectorizer()

X = vectorizer.fit\_transform(documents)

print(vectorizer.fit\_transform(documents))

print("Vocabulary:", vectorizer.get\_feature\_names\_out())

print("Document-term matrix:\n", X.toarray())

**Output:**

(0, 4) 1

(0, 0) 1

(0, 3) 1

(1, 4) 2

(1, 0) 1

(1, 3) 1

(1, 2) 1

(1, 1) 1

(2, 4) 2

(2, 0) 1

(2, 1) 1

(2, 5) 1

Vocabulary: ['cat' 'hat' 'in' 'sat' 'the' 'with']

Document-term matrix:

[[1 0 0 1 1 0]

[1 1 1 1 2 0]

[1 1 0 0 2 1]]

## Problem with Bag of words:

**Common Problems with Bag of Words:**

1. **Loss of Context**: BoW ignores the order of words, which can lead to loss of context. For example, "cat bites dog" and "dog bites cat" would have the same representation.
2. **High Dimensionality**: Large vocabularies can lead to very high-dimensional vectors, making the model computationally expensive.
3. **Sparse Data**: Most entries in the documentterm matrix are zeros, leading to a sparse matrix which can be inefficient.
4. **Inability to Handle Synonyms**: Different words with similar meanings (e.g., "happy" and "joyful") are treated as completely separate tokens.

# 2. TF-IDF (Term Frequency-Inverse Document Frequency)

## Concept:

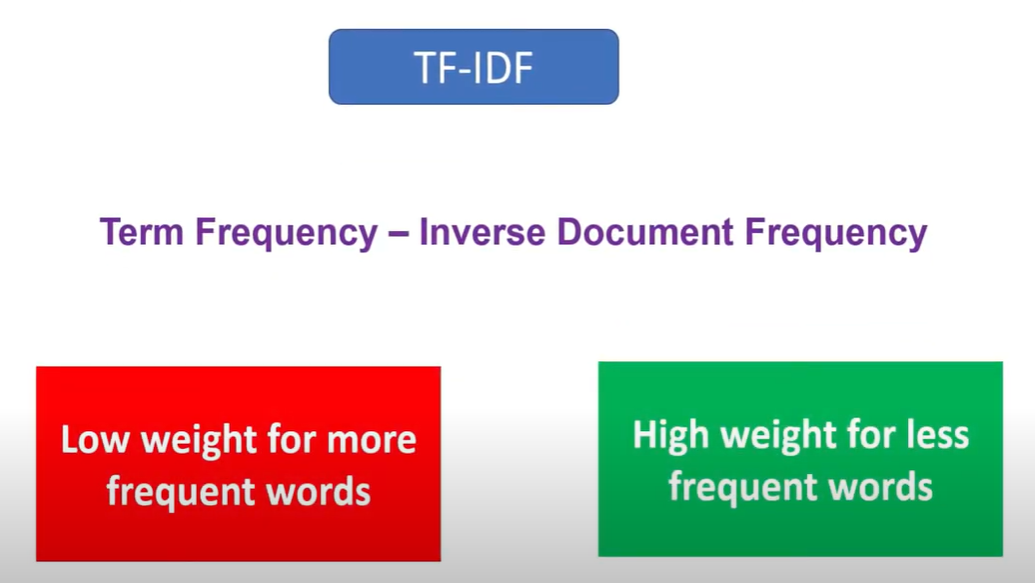
## 

TF-IDF is an extension of the Bag of Words model that accounts for the importance of words. It balances the frequency of a word in a document with its rarity across all documents.

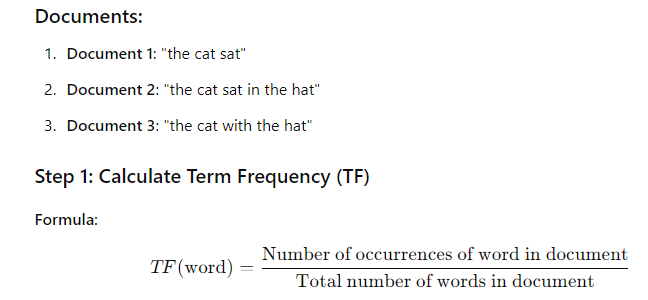
**TF-IDF (Term Frequency-Inverse Document Frequency)** is a statistical measure used in natural language processing (NLP) and information retrieval to evaluate the importance of a word in a document relative to a collection of documents (corpus). Unlike the **Bag of Words (BoW)** model, which only considers raw word counts, TF-IDF accounts for how often a word appears in a document *and* how rare or common the word is across the entire corpus.

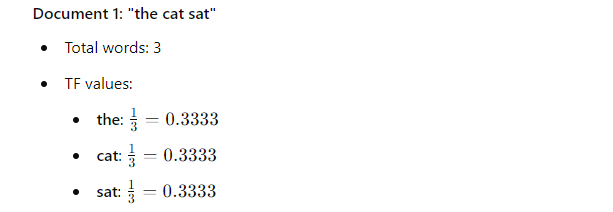
**Key Concepts:**

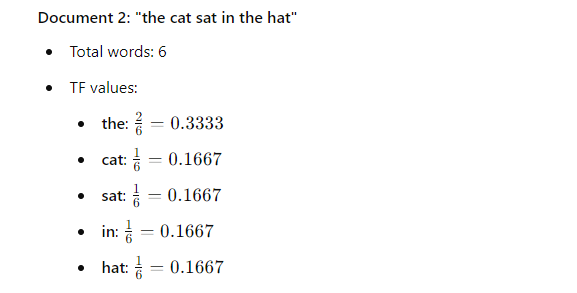
1. **Term Frequency (TF)**: Measures how frequently a word occurs in a document.
2. **Inverse Document Frequency (IDF)**: Measures how important a word is by considering how often it appears across all documents.
3. **TF-IDF Score**: The product of TF and IDF. Words that are frequent in a specific document but rare across the corpus will have high TF-IDF scores.

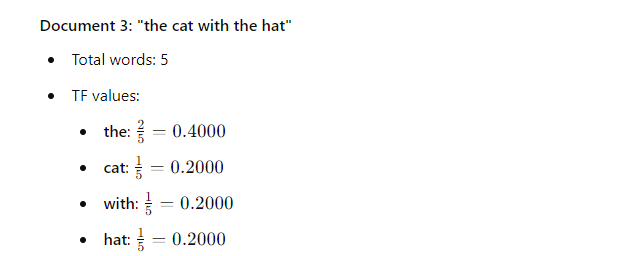


EXAMPLE:

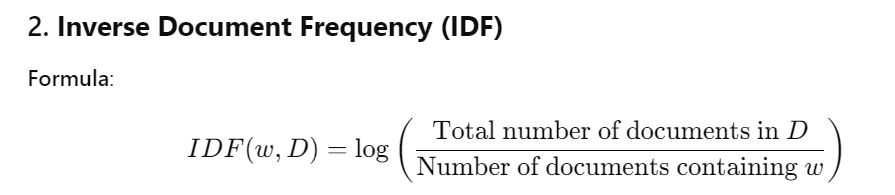




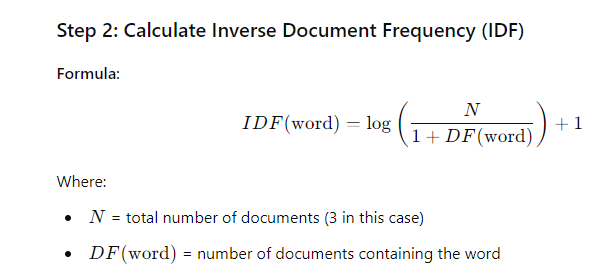


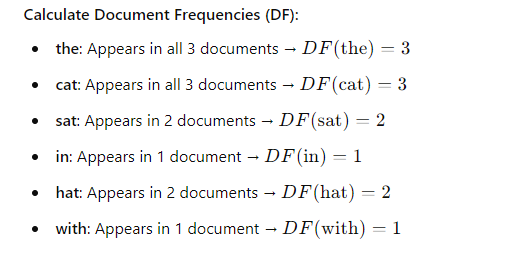


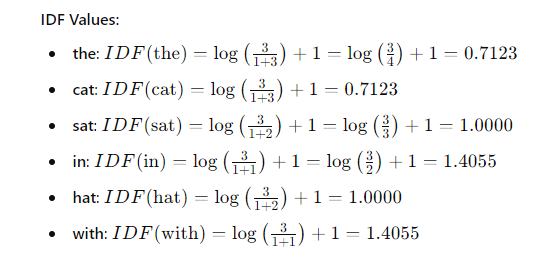
NOTE:

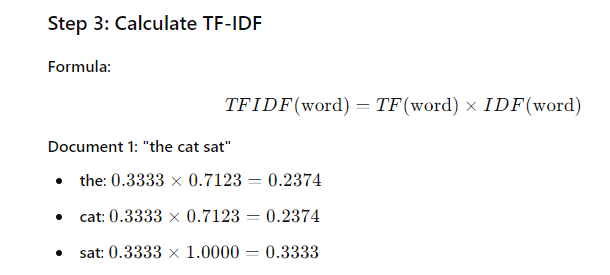


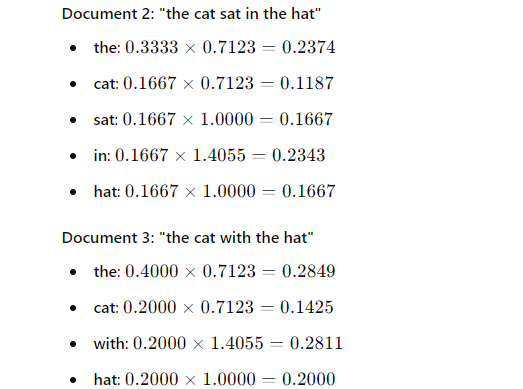
When we use above IDF formula we cannot handle the zero probability so we are using lathe smoothing technique to handle zero probability .This will make TF-IDF score accurate.











## Code :

import numpy as np

import pandas as pd

from collections import defaultdict

from sklearn.feature\_extraction.text import TfidfVectorizer

# Sample documents

documents = [

    "the cat sat",

    "the cat sat in the hat",

    "the cat with the hat"

]

# Step 1: Calculate Term Frequency (TF)

def calculate\_tf(documents):

    tf = []

    for doc in documents:

        words = doc.split()

        total\_words = len(words)

        tf\_dict = defaultdict(float)

        for word in words:

            tf\_dict[word] += 1 / total\_words

        tf.append(tf\_dict)

    return tf

# Step 2: Calculate Inverse Document Frequency (IDF)

def calculate\_idf(documents):

    n = len(documents)

    df = defaultdict(int)

    for doc in documents:

        unique\_words = set(doc.split())

        for word in unique\_words:

            df[word] += 1

    idf = {}

    for word, count in df.items():

        idf[word] = np.log(n / (1 + count)) + 1

    return idf

# Step 3: Calculate TF-IDF

def calculate\_tfidf(tf, idf):

    tfidf = []

    for tf\_dict in tf:

        tfidf\_dict = {word: tf\_dict[word] \* idf[word] for word in tf\_dict}

        tfidf.append(tfidf\_dict)

    return tfidf

# Execute the calculations

tf = calculate\_tf(documents)

idf = calculate\_idf(documents)

tfidf = calculate\_tfidf(tf, idf)

# Display results

print("Term Frequency (TF):")

for i, tf\_dict in enumerate(tf):

    print(f"Document {i+1}: {dict(tf\_dict)}")

print("\nInverse Document Frequency (IDF):")

for word, value in idf.items():

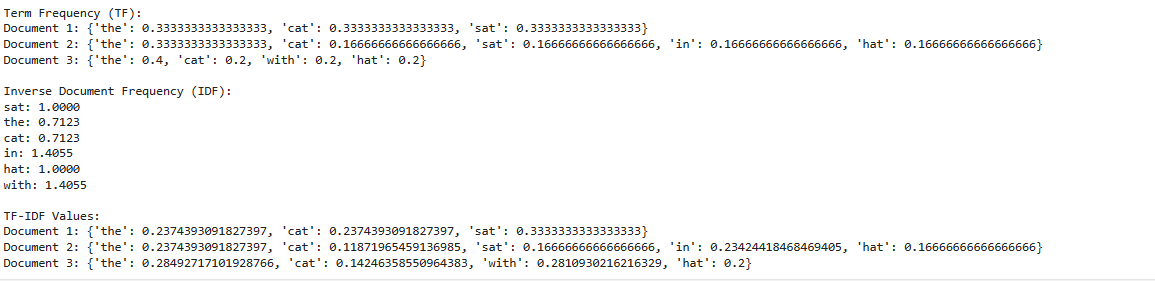
    print(f"{word}: {value:.4f}")

print("\nTF-IDF Values:")

for i, tfidf\_dict in enumerate(tfidf):

    print(f"Document {i+1}: {dict(tfidf\_dict)}")

OUTPUT:



Code for the Text Representation :

# Import necessary libraries  
from sklearn.feature\_extraction.text import CountVectorizer, TfidfVectorizer  
  
# Define the text corpus  
corpus = [  
 "Text representation is essential in NLP.",  
 "Bag of Words and TF-IDF are common techniques.",  
 "We are learning text representation techniques."  
]  
  
# Bag of Words Representation  
bow\_vectorizer = CountVectorizer()  
X\_bow = bow\_vectorizer.fit\_transform(corpus)

print("Vocabulary:", bow\_vectorizer.get\_feature\_names\_out())

print("Bag of Words Representation:\n", X\_bow.toarray())  
  
# TF-IDF Representation  
tfidf\_vectorizer = TfidfVectorizer()  
X\_tfidf = tfidf\_vectorizer.fit\_transform(corpus)  
print("Vocabulary:", tfidf\_vectorizer.get\_feature\_names\_out())

print("TF-IDF Representation:\n", X\_tfidf.toarray())  
  
Outputs:

Vocabulary: ['and' 'are' 'bag' 'common' 'essential' 'idf' 'in' 'is' 'learning' 'nlp'

'of' 'representation' 'techniques' 'text' 'tf' 'we' 'words']

Bag of Words Representation:

[[0 0 0 0 1 0 1 1 0 1 0 1 0 1 0 0 0]

[1 1 1 1 0 1 0 0 0 0 1 0 1 0 1 0 1]

[0 1 0 0 0 0 0 0 1 0 0 1 1 1 0 1 0]]

Vocabulary: ['and' 'are' 'bag' 'common' 'essential' 'idf' 'in' 'is' 'learning' 'nlp'

'of' 'representation' 'techniques' 'text' 'tf' 'we' 'words']

TF-IDF Representation:

[[0. 0. 0. 0. 0.44036207 0.

0.44036207 0.44036207 0. 0.44036207 0. 0.3349067

0. 0.3349067 0. 0. 0. ]

[0.35013871 0.26628951 0.35013871 0.35013871 0. 0.35013871

0. 0. 0. 0. 0.35013871 0.

0.26628951 0. 0.35013871 0. 0.35013871]

[0. 0.36617957 0. 0. 0. 0.

0. 0. 0.48148213 0. 0. 0.36617957

0.36617957 0.36617957 0. 0.48148213 0. ]]