Vector and Matrix Operations using Lists and Arrays
Vector Operations

- List can be used for representing vectors (1-D array).
- Addition of vectors

\[
V1 = \begin{bmatrix} 4 & 1 & 2 \end{bmatrix} \quad V2 = \begin{bmatrix} 3 & 5 & 7 \end{bmatrix} \\
V3 = V1 + V2 = \begin{bmatrix} 7 & 6 & 9 \end{bmatrix}
\]

```python
def add(V1, V2):
    V3 = [0 for i in range(len(V1))]
    for i in range(len(V1)):
        V3[i] = V1[i] + V2[i]
    return V3
```
Vector Operations

• Dot (scalar) product of vectors

\[ V_1 = \begin{pmatrix} 4 \\ 1 \\ 2 \end{pmatrix}, \quad V_2 = \begin{pmatrix} 3 \\ 5 \\ 7 \end{pmatrix} \]

\[ V_3 = V_1 \cdot V_2 = 31 \]

```python
def dotproduct(V1, V2):
    sum = 0
    for i in range(len(V1)):
        sum += V1[i] * V2[i]
    return sum
```
Vector Operations

• Cross product of vectors

\[ V1 = \begin{bmatrix} 4 & 1 & 2 \end{bmatrix} \quad V2 = \begin{bmatrix} 3 & 5 & 7 \end{bmatrix} \]

\[ V3 = V1 \times V2 = \begin{bmatrix} -3 & -22 & 17 \end{bmatrix} \]

```
def crossproduct(V1, V2):
    V3 = [0 for i in range(len(V1))]
    return V3
```
Matrix Operations

• Matrix can be represented as list of list
  For example a 3x3 matrix may be represented as
  \[
  \begin{bmatrix}
  a_1 & a_2 & a_3 \\
  b_1 & b_2 & b_3 \\
  c_1 & c_2 & c_3
  \end{bmatrix}
  \]

  Row 1  Row 2  Row 3

  A nxm matrix may be represented as
  \[
  \begin{bmatrix}
  a_{11} & a_{12} & \ldots & a_{1m} \\
  a_{21} & a_{22} & \ldots & a_{2m} \\
  \vdots & \vdots & \ddots & \vdots \\
  a_{n1} & a_{n2} & \ldots & a_{nm}
  \end{bmatrix}
  \]

  Row 1  Row 2  Row n
Matrix Multiplication

### Specification

\[ C_{i,j} = \sum_{k=0}^{n-1} A_{i,k} B_{k,j} \]

### Implementation

```python
C[i][j] = 0
for k in range(4):
    C[i][j] += A[i][k] * B[k][j]
```

Dot product approach

Courtesy Prof PR Panda CSE Department IIT Delhi
Matrix Multiplication

\[
\begin{array}{c|c|c|c|c}
0 & 1 & 2 & 3 \\
\hline
i,0 & i,1 & i,2 & i,3 \\
\end{array}
\times
\begin{array}{c|c|c|c|c}
0 & 1 & 2 & 3 \\
\hline
0,j & 1,j & 2,j & 3,j \\
\end{array}
= \begin{array}{c|c|c|c|c}
0 & 1 & 2 & 3 \\
\hline
i,j \\
\end{array}
\]

def mult(A,B,n,p,m):
    C=[[0 for col in range(m)] for row in range(n)]
    for i in range(n):
        for j in range(m):
            for k in range(p):
                C[i][j]+=A[i][k]*B[k][j]
    return C

Courtesy Prof PR Panda CSE Department IIT Dehi
def mult(A, B, n, p, m):
    C = [[0 for col in range(m)] for row in range(n)]
    for i in range(n):
        for j in range(m):
            for k in range(p):
                C[i][j] += A[i][k] * B[k][j]
    return C

Better Version!
Less memory access time

def mult(A, B, n, p, m):
    C = [[0 for col in range(m)] for row in range(n)]
    for i in range(n):
        for j in range(m):
            t = 0
            for k in range(p):
                t += A[i][k] * B[k][j]
            C[i][j] = t
    return C
Matrix Transpose

def transpose(A, n, m):
    AT = [[0 for col in range(n)] for row in range(m)]
    for i in range(n):
        for j in range(m):
            AT[j][i] = A[i][j]
    return AT
NumPy Package

- NumPy is a Python package for numerical computation
- NumPy core data type is array
- NumPy must be installed if to be used

```python
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
a * b
```
```
array([ 4, 10, 18])
```
NumPy Package

• NumPy is a Python package for numerical computation
• NumPy core data type is array
• NumPy must be installed if to be used

```python
import numpy as np
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
np.dot(a,b)
32
```