# SEEE1022 INTRODUCTION TO SCIENTIFIC PROGRAMMING

## CH9
## Matrix Algebra

Dr. Mohd Saiful Azimi Mahmud (azimi@utm.my)
P19a-04-03-30, School of Electrical Engineering, UTM

UTM
UNIVERSITI TEKNOLOGI MALAYSIA
www.utm.my
innovative • entrepreneurial • global
univteknologimalaysia
utm_my
utmofficial

# OBJECTIVES

- To introduce several matrix algebra functions to solve engineering problems.
- To understand on how to use matrix algebra function such:
  i. Dot product
  ii. Cross product
  iii. Inverse
  iv. Determinant

- To introduce other matrix function that can be useful in solving engineering problems.

# MATRIX ALGEBRA

# MATRIX ALGEBRA

## INTRODUCTION

- MATLAB was originally written to provide an easy-to-use interface to professionally developed numerical linear algebra subroutines.
- It offers a wide range of valuable matrix algebra functions
- Note that while MATLAB supports n-dimensional arrays, matrix algebra is defined only for 2-D arrays — that is, vectors and matrices.
- Matrix algebra is used extensively in engineering applications

- **The difference between an array and a matrix:**
  i. Most engineers use the two terms interchangeably.
  ii. The only time you need to be concerned about the difference is when you perform matrix algebra calculations.

# MATRIX ALGEBRA

## SOLUTION

1) Dot product (math symbol '·')
2) Cross product (math symbol '×')
3) Inverse
4) Determinants
5) Linear equation

## DOT PRODUCTS

- The dot product is the sum of the results when you multiply two vectors of the same length together, element by element.

EXAMPLE 1

$$\boldsymbol{u} = [3\ 2\ 6\ 4], \qquad \boldsymbol{v} = [4\ 2\ 3\ 1]$$

$$\boldsymbol{u} \cdot \boldsymbol{v} = [3 \quad 2 \quad 6 \quad 4] \cdot \begin{bmatrix} 4 \\ 3 \\ 2 \\ 1 \end{bmatrix} = (3 \times 4) + (2 \times 3) + (6 \times 2) + (4 \times 1) = 34$$

- In Matlab, either matrix multiplication operator `*` or function `dot()` can be used to perform the dot product.

## USING *OPERATOR AS DOT PRODUCT

- When performing matrix multiplication, the number of columns in the first matrix must equals to the number of rows in second matrix.
- To apply the matrix multiplication operator * as a dot product of two vectors, set the vectors as below

Length of the vectors

$$(1 \times n) \;*\; (n \times 1)$$

The resulting matrix will have $1 \times 1$ dimensions (the dot product)

- Note that the first vector is a row vector and the second vector is a column vector.

## ANGLE OF TWO VECTORS

EXAMPLE 2

Formula to compute angle between two vectors $\vec{u}$ and $\vec{v}$ is as below where the numerator of the inverse cosine is a dot product between the two vectors while $||\vec{u}||$ and $||\vec{v}||$ are the magnitude of the vectors:

$$\theta = \cos^{-1} \frac{\vec{u}.\vec{v}}{||\vec{u}||.||\vec{v}||}$$

$$||\vec{u}|| = \sqrt[2]{u_1^2 + u_2^2 + \cdots + u_N^2} = (\vec{u}.\vec{u}')^{\frac{1}{2}}$$

Write a MATLAB function to compute angle between the following two vectors:

$$u = [3\ 4\ 1]$$

$$v = [1\ 1\ 1]$$

# ANGLE OF TWO VECTORS

## EXAMPLE 2

MATLAB code for Example 2

```
function theta = vecAngle(u,v)
uMag = sqrt(u*u'); %1st dot product
vMag = sqrt(v*v’); %2nd dot product
udotv = u*v';      %3rd dot product

theta = acos(udotv/(uMag*vMag));
thetadeg = theta*180/pi;

fprintf('\x3B8 = %.2f\x3C0 or %.2f\xB0\n',theta/pi,thetadeg)
```

```
>> u = [3 4 1];
>> v = [1 1 1];
>>
>> theta = vecAngle(u,v);
θ = 0.14π or 25.07°
```

The 1st and 2nd dot product applied a transposed to its second term to fulfil the size compatibility of $1\text{x}n \cdot n\text{x}1$

## * IS NOT ALWAYS A DOT PRODUCT

- Other than the $(1 \times n) * (n \times 1)$ vector size format, the * operator will not return a dot matrix.

EXAMPLE 3

```
>> u = [3 4 1];
>> v = [1 1 1];
>> y = u'*v
y =
     3     3     3
     4     4     4
     1     1     1
```

Dot product should return a scalar, not a matrix

- When the first vector is a column vector as the above example, it can be used as one of the solution to the vectorizing method covered in Chapter 4. Note that in Chapter 4, array operator is used for the vectorizing method.

## * IS NOT ALWAYS A DOT PRODUCT

### EXAMPLE 4

Lets recap Example 16 from Chapter 4 where a formula of compound interest `B` is evaluated for `3` values of `a` (`$100,$500,$800`) on `5` different total year of `n` (`2,4,6,8,10`) and `r=0.09`. Instead of the array operation, the array multiplication can be replaced with matrix multiplication as below.

```
a=[100 500 800];
n=[2 4 6 8 10];
r=0.09;

% Using array multiplication
[N,A] = meshgrid(n,a)
B = A.*(1+r).^N

% Using matrix multiplication
B = a'*(1+r).^n
```

- For the matrix multiplication method, it is done on $(3 \times 1) * (1 \times 5)$ size format, thus resulting a $3 \times 5$ matrix, similar to the array multiplication method.
- Since it only works for multiplication, the power operation is maintained with the array operation.
- In this case meshgrid is not needed.

## USING `dot()` FUNCTION

- Since * operator is not always a dot product, dot() function is a less confusing method to perform the dot product.

**EXAMPLE 5**

```
>> u = [3 4 1];
>> v = [1 1 1];
>> y = dot(u,v)
y =
     8
```

## CENTER OF MASS

### EXAMPLE 6

Table below shows the position of several components in x-y-z coordinate together with the component's mass.

| $i$ | Item | $x$ (meter) | $y$ (meter) | $z$ (meter) | Mass, $m$ (gram) |
|---|---|---|---|---|---|
| 1 | Bolt | 0.1 | 2 | 3 | 3.50 |
| 2 | screw | 1 | 1 | 1 | 1.50 |
| 3 | nut | 1.5 | 0.2 | 0.5 | 0.79 |
| 4 | bracket | 2 | 2 | 4 | 1.75 |

Below is the formula on how to compute the center of mass position on the above components.

$$\bar{x} = \frac{\sum x_i m_i}{\sum m_i} = \frac{\boldsymbol{x} \cdot \boldsymbol{m}}{\sum m_i}, \qquad \bar{y} = \frac{\boldsymbol{y} \cdot \boldsymbol{m}}{\sum m_i}, \qquad \bar{z} = \frac{\boldsymbol{z} \cdot \boldsymbol{m}}{\sum m_i}$$

# MATRIX ALGEBRA

## CENTER OF MASS

EXAMPLE 6

- From the equation, the $\boldsymbol{x} \cdot \boldsymbol{m}$, $\boldsymbol{y} \cdot \boldsymbol{m}$ and $\boldsymbol{z} \cdot \boldsymbol{m}$ is a dot product operation.
- Lets rewrite the three dot product equations in matrix form by setting vector $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ into a single matrix $P$:

$$D = P.\boldsymbol{m} = \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ z_1 & z_2 & z_3 & z_4 \end{bmatrix} \cdot \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ m_4 \end{bmatrix} = \begin{bmatrix} 0.1 & 1 & 1.5 & 2 \\ 2 & 1 & 0.2 & 2 \\ 3 & 1 & 0.5 & 4 \end{bmatrix} \cdot \begin{bmatrix} 3.5 \\ 1.5 \\ 0.79 \\ 1.75 \end{bmatrix} = \begin{bmatrix} 6.53 \\ 12.16 \\ 19.40 \end{bmatrix}$$

- To get the center of mass, $D$ is divided by the sum of the mass.

$$\bar{P} = \frac{D}{\sum m_i} = \frac{[6.53 \quad 12.16 \quad 19.40]}{3.50 + 1.50 + 0.79 + 1.75} = \frac{[6.53 \quad 12.16 \quad 19.40]}{7.54} = [0.87 \quad 1.61 \quad 2.57]$$

## CENTER OF MASS

EXAMPLE 6

To implement the center of mass computation, we can either compute the $\bar{x}$, $\bar{y}$ and $\bar{z}$ separately or as a single matrix.

Below is the MATLAB code when $\bar{x}$, $\bar{y}$ and $\bar{z}$ are computed separately:

```
m = [3.5, 1.5, 0.79, 1.75];
x = [0.1,   1,   1.5, 2];
y = [  2,   1,   0.2, 2];
z = [  3,   1,   0.5, 4];
xbar = dot(x,m)/sum(m);
ybar = dot(y,m)/sum(m);
zbar = dot(z,m)/sum(m);

fprintf(['Center of mass:\nx = %.2f\n'...
         'y = %.2f\nz = %.2f\n'],xbar,ybar,zbar)
```

```
Center of mass:
x = 0.87
y = 1.61
z = 2.57
```

## CENTER OF MASS

### EXAMPLE 6

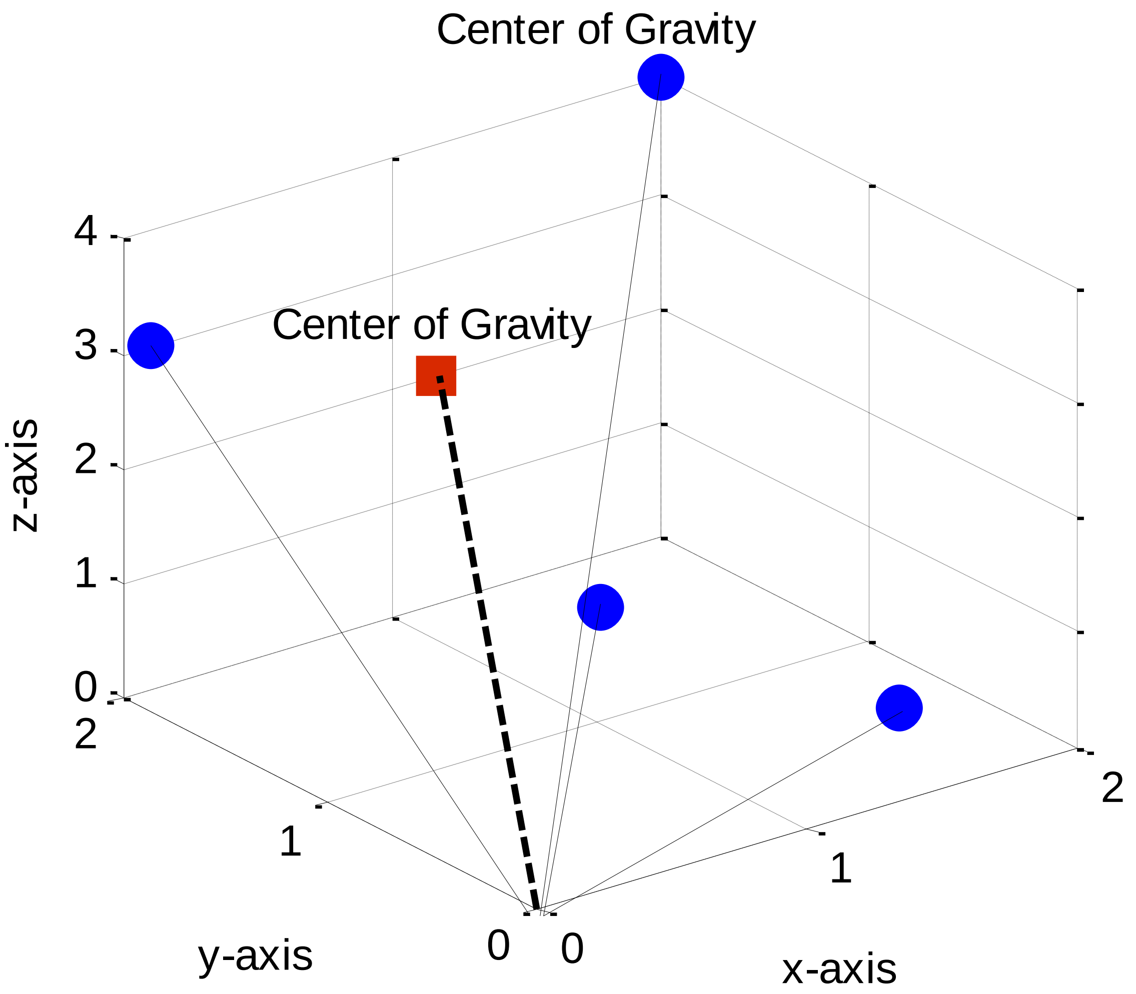
Below is the MATLAB code when $\bar{x}$, $\bar{y}$ and $\bar{z}$ are combined into single matrix.

```
m = [3.5, 1.5, 0.79, 1.75];
P = [0.1,    1,   1.5, 2
       2,    1,   0.2, 2
       3,    1,   0.5, 4];
Pbar = P*m'/sum(m);

fprintf(['Center of mass:\nx = %.2f\n'...
         'y = %.2f\nz = %.2f\n'],Pbar)
```

```
Center of mass:
x = 0.87
y = 1.61
z = 2.57
```

- Combining vector $\bar{x}$, $\bar{y}$ and $\bar{z}$ into single matrix means the three dot product are performed at one single code.
- * operator is a better option since `dot()` function require both inputs to have similar size.

## CENTER OF MASS



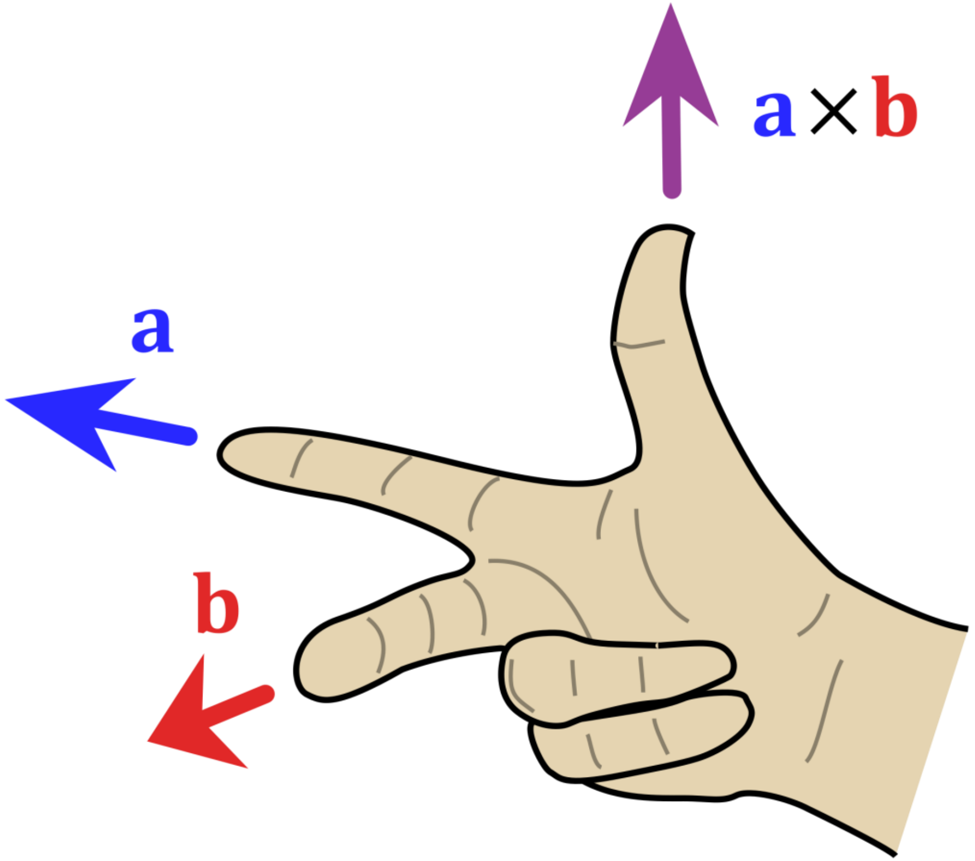

This plot was enhanced using the interactive plotting tools

## CROSS PRODUCT

- Given any two vector of $\vec{A}$ and $\vec{B}$, and $\theta$ as the angle between them, the cross product of the two vectors is:

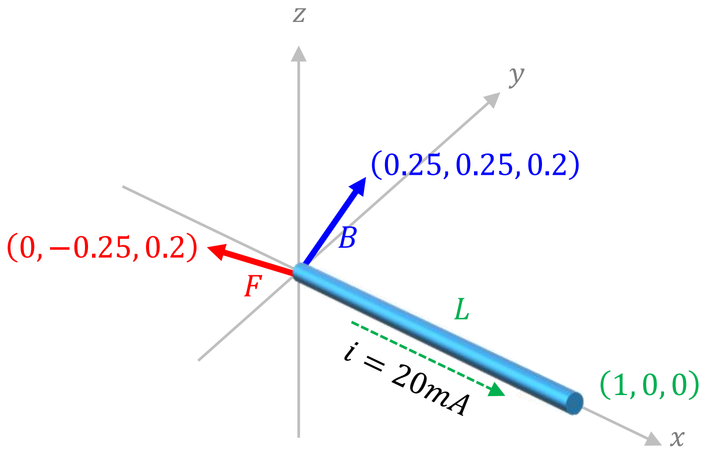
$$\vec{C} = \vec{A} \times \vec{B}$$
$$|C| = |A| \cdot |B| \sin(\theta)$$

- $\vec{C}$ is perpendicular to $\vec{A}$ and $\vec{B}$.
- The direction of $\vec{C}$ is given by the right hand rule.
- In MATLAB, the cross product can be computed using function `cross()`.



## MAGNETIC FORCE

EXAMPLE 7




When a wire of length $L$ carries a current $i$ through a magnetic field $B$, the magnetic force $|F|$ by the field on the wire is:

$$\vec{F} = i(\vec{L} \times \vec{B}) \qquad \text{(Eq. 1)}$$

$$|F| = i|\vec{L} \times \vec{B}| \qquad \text{(Eq. 2)}$$

$$|F| = i|L| \cdot |B| \sin(\theta) \qquad \text{(Eq. 3)}$$

## MAGNETIC FORCE

EXAMPLE 7

To compute the magnetic force $|F|$, we can either use cross product (Eq. 2) or dot product (Eq. 3). Below is the MATLAB code for both method where using cross product gives a simpler code:

```
L = [1 0 0];
B = [0.25 0.25 0.20];
i = 20e-3;

% Cross product method
LB = cross(L,B)
F = i*norm(LB)

% Dot product method
theta = acos(dot(L,B)/(norm(L)*norm(B)));
F = i*norm(L)*norm(B)*sin(theta);
```

`norm()` is a function to compute magnitude of a vector.

```
LB =
      0    -0.2000    0.2500
F =
      0.0064
```

## MATRIX INVERSE

- Matrix inverse properties:

    1) Inverse of a matrix $A$, though written mathematically as $A^{-1}$ is not equals to $1/A$.
    2) If an inverse of a scalar multiply with the scalar is equals to one $(a^{-1}a = 1)$, multiplication of an inverse matrix to the original matrix is an identity matrix.
    3) Only square matrix (of size $m \times m$) has an inverse matrix.
    4) Singular matrix does not has an inverse.

- MATLAB offers three approaches:

    1) The matrix inverse function, `inv()`.
    2) Raising a matrix to the -1 power, `A^-1`.
    3) Using left division operator '\' when multiply with a column vector. For example, $A^{-1}B$ where $B$ is a column vector can be written in MATLAB code as `A\B`.

## MATRIX INVERSE

### EXAMPLE 8

```
>> A = [2 4; 6 3]
A =
     2     4
     6     3

>> inv(A)
ans =
   -0.1667    0.2222
    0.3333   -0.1111

>> A^-1
ans =
   -0.1667    0.2222
    0.3333   -0.1111

>> A\[1 1]'
ans =
    0.0556
    0.2222
```

Both `inv(A)` and `A^-1` return the inverse of matrix A

Using left division operator is an optimized method in performing `inv(A)*[1 1]'`.

## DETERMINANT

- Determinant is a useful value that can be computed from the elements of a **square matrix**.
- The determinant of a matrix $A$ is denoted $\det(A)$, $det\ A$, or $|A|$.
- For a $2 \times 2$ matrix, the formula for the determinant is:

$$|A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

EXAMPLE 9

$$\begin{vmatrix} 4 & 2 \\ 2 & 8 \end{vmatrix} = (4)(8) - (2)(2) = 28$$

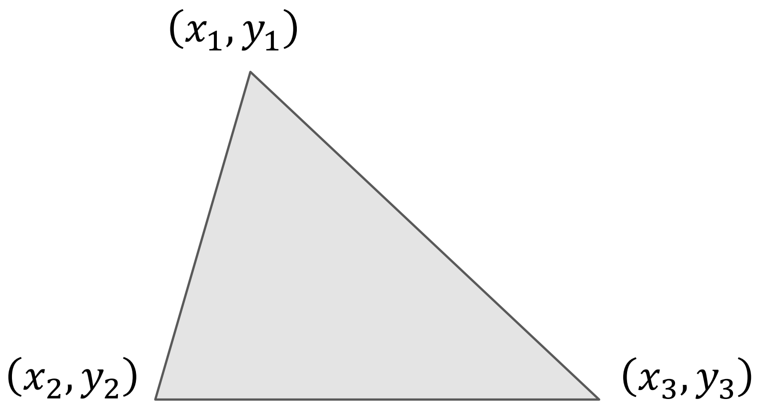
## CHECKING SINGULAR MATRIX

- Singular matrix is a matrix that does not has an inverse. A matrix is said to be a singular matrix if its determinant is equals to 0.

EXAMPLE 10

```
>> A = [1 2 3; 4 5 6; 1 2 3]
A =
     1     2     3
     4     5     6
     1     2     3

>> det(A)
ans =
     0

>> inv(A)
Warning: Matrix is singular to working precision.

ans =
   Inf   Inf   Inf
   Inf   Inf   Inf
   Inf   Inf   Inf
```

## AREA OF A TRIANGLE




Area of a triangle can be formulated using matrix determinant as below:

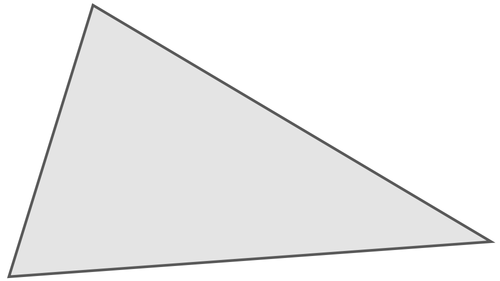
$$A = \left|\frac{1}{2}\det(T)\right|$$

Where $T$ is the matrix of cartesian coordinate of the three corners of the triangle. Thus matrix $T$ is set as below. The 1st column is the x-coordinate, 2nd column is the y-coordinate and the 3rd column is set equals to 1 to form a square matrix.

$$T = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}$$

## AREA OF TRIANGLE

EXAMPLE 11




```
>> T = [2 1 1; 3 7 1; 8 2 1]
T =
     2     1     1
     3     7     1
     8     2     1

>> A = abs(0.5*det(T))
A =
     17.5000
```

## SETS OF LINEAR EQUATIONS

- One of the most common linear algebra problems is finding the solution of a linear set of equations. For example, consider the set of equations.

$$4x_1 + 5x_2 + 6x_3 = 232 \qquad (1)$$
$$23x_1 + 2x_2 + 84x_3 = 401 \qquad (2)$$
$$-3x_1 - 5x_2 + 1x_3 = 198 \qquad (3)$$

- These linear equations can be written in matrix form as below

$$\underset{A}{\begin{bmatrix} 4 & 5 & 6 \\ 23 & 2 & 84 \\ -3 & -5 & 1 \end{bmatrix}} \cdot \underset{x}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}} = \underset{y}{\begin{bmatrix} 232 \\ 401 \\ 198 \end{bmatrix}}$$

- Then $x$ can be solved by multiplying inverse of matrix $A$ with $y$ as follow:

$$x = A^{-1}y$$

In MATLAB, the preferable solution is found by using the matrix left-division operator, `x = A\y.`

# MATRIX ALGEBRA

## LINEAR EQUATION

EXAMPLE 12

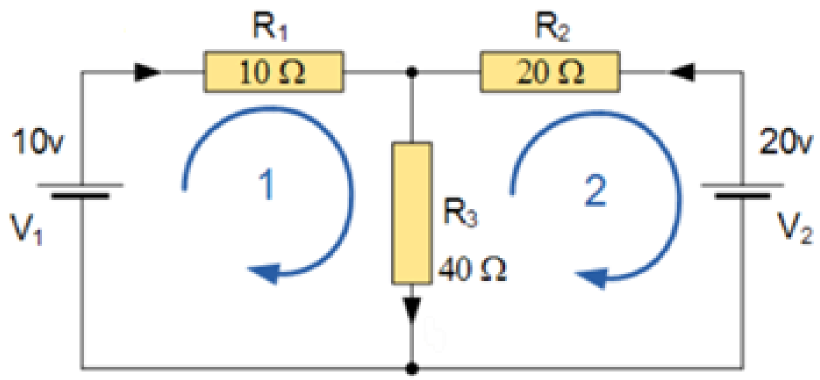
MATLAB code for previous slide

```
>> A = [4 5 6; 23 2 84; -3 -5 1]
A =
      4      5       6
     23      2      84
     -3     -5       1

>> y = [232;401;198]
y =
     232
     401
     198

>> x = A\y
x =
    -482.8534
     276.1935
     130.4076
```

# MATRIX ALGEBRA

## CIRCUIT ANALYSIS

EXAMPLE 13




In analysing the above circuit, current on loop 1 and loop 2 can be computed based on two equation:

$$(10+40)I_1 - 40I_2 = 10 \qquad (Eq.1)$$
$$-40I_1 + (20+40)I_2 = -20 \qquad (Eq.2)$$

Using matrix algebra, $I_1$ and $I_2$ can be simply solved as below:

$$\begin{bmatrix} I_1 \\ I_2 \end{bmatrix} = \begin{bmatrix} 50 & -40 \\ -40 & 60 \end{bmatrix}^{-1} \begin{bmatrix} 10 \\ -20 \end{bmatrix}$$

## CIRCUIT ANALYSIS

EXAMPLE 13

MATLAB code for Example 13

```
>> R = [50 -40; -40 60]
R =
    50   -40
   -40    60

>> V = [10;-20]
V =
    10
   -20

>> I = R\V
I =
   -0.1429
   -0.4286

>> I = inv(R)*V
I =
   -0.1429
   -0.4286
```

Example of using function `inv()` to solve $I$.
However this is not preferable in MATLAB.
Left division is the most optimized method.

# USEFUL MATRIX FUNCTION

# USEFUL MATRIX FUNCTION

| Function | Description |
|---|---|
| Create and Combine Array | |
| `cat` | Concatenate arrays along specified dimension |
| `horzcat` | Concatenate arrays horizontally |
| `vertcat` | Concatenate arrays vertically |
| `repmat` | Repeat copies of array |
| Reshape and Rearrange | |
| `sort` | Sort array element |
| `sortrow` | Sort rows of matrix |
| `flip` | Flip order of element |
| `trace` | Compute the sum of the elements on the main diagonal. |
| Special Matrices | |
| `ones` | Create array of all ones. |
| `zeros` | Create array of all zeros. |
| `eye` | Identity matrix |
| `magic` | Magic matrix |

# USEFUL MATRIX FUNCTION

## TRACE

EXAMPLE 14

```
>> A = [1 2 3; 4 5 6; 7 8 9]

A =
     1     2     3
     4     5     6
     7     8     9

>> B = trace(A)

B =
    15
```

# USEFUL MATRIX FUNCTION

## EYE

- Lets consider

  $F(\boldsymbol{x}) = \boldsymbol{x} + A\boldsymbol{x} + B\boldsymbol{x}$

  where A and B are 2x2 matrices, and $\boldsymbol{x}$ is a 2x1 vector.

- Instead of having two multiplication, the equation can be rearranged to have single multiplication as below where $I$ is a 2x2 identity matrix. Having less multiplication will make the operation faster.

$$F(\boldsymbol{x}) = (I + A + B)\boldsymbol{x}$$

- Above equation will return wrong result if scalar value 1 is used instead of the identity matrix.
- `eye()` is a MATLAB function to create the identity matrix.

# USEFUL MATRIX FUNCTION

## EYE

EXAMPLE 15

```
>> A = [1 2;3 4];
>> B = [2 3;5 6];
>> X = [2 3];

>> F = X + X*A + X*B
F =
    32    43

>> I = eye(2)
I =
     1     0
     0     1

>> F = X*(I + A + B)
F =
    32    43

>> F = X*(1 + A + B)
F =
    35    45
```

The answer is wrong when scalar value 1 is used instead of the identity matrix.

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

univteknologimalaysia

utm_my

utmofficial

# Thank You

www.utm.my

innovative • entrepreneurial • global