Welcome to this spoken tutorial on Matrix Operations using Scilab.

To practice this tutorial, you need to have Scilab installed on your system. Open the scilab console window.

Let us start by defining a vector. This can be done in two ways: first one is using spaces

```
-->p = [1 2 3]

 p  =

    1.    2.    3.
```

or using commas.

```
-->q = [2, 3, 4]

 q  =

    2.    3.    4.
```

We can find the length of a vector as follows:

```
-->length(p)

 ans  =

    3.
```

We can work with vectors, just as we do in Maths classes. We can add them:

```
-->p + q

 ans  =

    3.    5.    7.
```

We can subtract

```
-->q - p

 ans  =

    1.    1.    1.
```

and so on.

Transpose of a vector can be found using apostrophe.

```
-->p'

 ans  =

    1.
```

2.

3.

This is also known as a prime or a single quote. On my keyboard, it is next to the enter or return key.

We can calculate p-transpose (times)* q:

```
-->p'*q
 ans  =
    2.    3.    4.
    4.    6.    8.
    6.    9.    12.
```

The command p times q-transpose gives a scalar:

```
-->p*q'
 ans  =
    20.
```

I suggest that you pause at this point and check these calculations. I would want you to do this at regular intervals.

When defining matrices, the semicolon is used for defining the next row. For example, define a 2 by 3 matrix P by typing

```
-->P = [1 2 3;4 5 6]
 P  =
    1.    2.    3.
    4.    5.    6.
```

Note that this P is in upper case. In Scilab, upper case variables are different from lower case variables. Would you want to check what small p has at this point?

The size of a matrix can be obtained by using the size command

```
-->size(P)
 ans  =
    2.    3.
```

answer is 2 , 3

We can assign these values to variables as follows:

-->[row, col] = size(P)

col =

3.

row =

2.

The transpose command works for matrices as well:

-->P'

ans =

1. 4.

2. 5.

3. 6.

as given here

Let us define another 2 by 3 matrix Q:

-->Q = [1 5 3;2 4 8]

Q =

1. 5. 3.

2. 4. 8.

Let us also recall P once more:

We can carry out calculations involving P and Q, just as we do in mathematics.

For example, let us calculate:

-->E = 2*P + 3*Q

E =

5. 19. 15.

14. 22. 36.

You may want to verify whether these calculations are correct.

We can address the elements of matrices easily. For example, if we want to access the 2,2 element, we give:

```
-->E(2,2)

ans  =

    22
```

It is easy to address rows and columns of matrices. For example, the first row of Q is obtained using the following command:

```
-->Efr = E(1,:)

Efr  =

   5.    19.    15.
```

You see that we get the first row.

The semicolon operator says, take everything. Here, it says that we have to take all columns, namely, three columns. Let us recall E. The second to third columns of E are obtained using

```
 -->Est = E(:,2,3)
```

I made a mistake

so 2 to 3 is given by this colon operator

```
-->Est = E(:,2:3)

Est  =

    19.    15.

   22.    36.
```

Ok you can see that. Let recall what E is.

You can see that.

The first colon says take everything.

2 to 3 says take from 2 to 3 onwards

I have a question for you. What will E(:,:) give?

I will not give this command.

I would want you to try this out.

I would want you to explain why it happens.

One of the important operations carried out on matrices is elementary row operations. In this, we have to make entries below a number to be zero. This can be done easily in Scilab.

Let us recall P.

Suppose that we want to make the (2,1) entry zero by elementary row operation. That is, we would want to multiply the first row by 4 and subtract it from the second row:

```
-->P(2,:) = P(2,:) - 4*P(1,:)

 P  =

    1.    2.    3.

    0.  - 3.  - 6.
```

Would you want to pause and check out this calculation?

This procedure can be extended to larger systems and to elementary column operations.

We can easily append rows and columns to matrices. For example, if we want to append a row containing [5 5 -2], we do the following:

```
-->T = [P; (go to the next row) [5 5 -2]]
```

(this row is over this operation is over)

```
 T  =

    1.    2.    3.

    0.  - 3.  - 6.

    5.    5.  - 2.
```

The semicolon after P says that the numbers coming after should go to the next row.

We use matrix notation while solving equations. Suppose that we want to solve this matrix equation:

<math>

 x_1 + 2x_2 - x_3 = 1

</math>

<math>

 -2x_1 - 6x_2 + 4x_3 = -2

</math>

<math>

 -x_1 - 3x_2 + 3x_3 = 1

</math>

This can be written as follows Ax = b

The solution is given as Inverse A times b

Lets verify whether this is correct

A is given as

```
 -->A = [1 2 -1;-2 -6 4;-1 -3 3]
 A  =
   1.    2.  - 1.
 - 2.  - 6.    4.
 - 1.  - 3.    3.
  -->b = [1;-2;1]
 b  =
   1.
 - 2.
   1.
```

This is the same system we saw here

Let get back here

We can calculate x using

```
 -->x = inv(A)*b
 x  =
 - 1.
   2.
   2.
```

we get this result

We can also solve it in scilab with A reverse slash b

It is as if this A is taken from this side over here

so it comes to the left of b and because A divides b you put a slash in this manner

Let us do it in Scilab

```
 -->x = A\b
```

```
 x  =

  - 1.

    2.

    2.
```

We get the same result

The first method involves two steps: calculation of inverse of A and then multiplication with b.

A\b just tells Scilab to solve this system. This is done in one step. As a result, A\b is generally more efficient.

We can verify the correctness of the solution by back substitution, that is, by calculating Ax-b:

```
 -->A*x-b

 ans  =

     0.

    0.

    0.
```

We see that the solution is indeed correct. It is possible that in some systems we may not get zero exactly. But we will get a number close to zero, say, of the order of 10^{-16} or something like that.

For square matrices, we can calculate the determinants:

Let us recall A first:

Lets calculate the determinant of A

```
 -->det(A)

 ans  =

  - 2.
```

We can calculate the square or cube of a square matrix A by simply typing A^2 or A^3. Here we use the carat symbol. In my keyboard, it is obtained by shift 6.

One can calculate the eigenvalues of a square matrix by the command spec, which is a short form for spectrum:

as i do here

```
 -->spec(A)
```

ans =

- 3.7448261

0.3959319

1.3488942

It is possible to calculate the eigenvectors also using the spec command. You can find the syntax for this by giving this command

-->help spec

I will not give this command, however.

Using Scilab we can create special matrices:

For example we can create a matrix of zeros with 3 rows and 4 columns using this command

-->zeros(3,4)

ans =

0. 0. 0. 0.

0. 0. 0. 0.

0. 0. 0. 0.

A matrix of all ones can be created with ones command as follows

-->ones(2,4)

ans =

1. 1. 1. 1.

1. 1. 1. 1.

It is easy to create identity matrices as well:

-->eye(4,4)

ans =

1. 0. 0. 0.

0. 1. 0. 0.

0. 0. 1. 0.

0. 0. 0. 1.

This is a 4 by 4 identity matrix.

We often need matrices consisting of pseudo random numbers. These can be generated by using the rand command as follows

```
-->p=rand(2,3)

 p =

0.2113249 0.0002211 0.6653811

0.7560439 0.3303271 0.6283918
```

See help rand for options regarding the distributions and seeds.

This brings us to the end of spoken tutorial on Matrix Operations using Scilab.

We also have several other spoken tutorial on Scilab at this time.

These are listed here.

Scilab Effort in India is co-ordinated through this website.

scilab.in

There are some interesting projects one of them is the Textbook project

That codes worked out examples of standard textbooks using scilab

The link project allows users to link known scilab documents and to rank them

We also help organize Scilab Workshops

We have two mailing lists one for announce and another one for discuss.

We invite your participation in all our activities

Lets get back to spoken tutorials

The spoken part will be available in various Indian Languages as well.

These are available at spoken-tutorial.org

These tutorial form a part of Level 0 training in Scilab.

These tutorials are available absolutely free of cost.

We wish to cover many FOSS systems through this route.

We welcome your feedback on these.

We also welcome your participation

On writing the outline for the software.

To write the original scripts.

To record the spoken tutorial.

To translate the script into various Indian Languages.

To dub the audio in Indian Languages using the script.

To review and give your feedback on all of the above.

We welcome you to conduct workshops using these spoken tutorials.

We also invite you to conduct efficacy studies on Spoken tutorials.

We are also looking for experts who can give technology support for audio, video, automatic translation, etc.

We have funding for all these activities

The Spoken Tutorials are part of the Talk to a Teacher project, supported by the National Mission on Education through ICT abbreviated as NMEICT given by MHRD government of India. More information on the same is available at this website http://spoken-tutorial.org/NMEICT-Intro.

Thanks for joining us. This is Kannan Moudgalya signing off. Goodbye.