GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2003

Lab #2: Introduction to Complex Exponentials

Date: 21–27 Jan. 2003

You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time. You **MUST** complete the online Pre-lab exercise on Web-CT at the beginning of your lab session during the period 21-Jan to 27-Jan. You can use MATLAB or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Lab exercise. The Pre-Lab exercise for this this lab also includes some questions about concepts from the *previous* report.

The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 4 as the lab report for this lab. More information on the lab report format can be found on Web-CT under the ‘Information” link. You are asked to **label** the axes of your plots and include a title for every plot. In order to reduce missing plots, include your plot as a figure *embedded* within your report. For more information on how to include figures and plots from MATLAB in your report file, consult the “information” link on Web-CT. If you still do not know how to do so, ask your TA.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.

The lab report and verification will be graded out of 100 points. The Pre-Post-Lab questions will be graded separately.

The lab report will be **due during the period 28-Jan. through 3-Feb. at the start of your lab.**

PRINTING BUDGET: For the printers in the ECE labs, you have a quota. Please limit your printing to essential items for the labs. If you need to print lecture slides and other large documents, use the central (OIT) printing facilities.

---

1 Introduction and Overview

The goal of this laboratory is to gain familiarity with complex numbers and their use in representing sinusoidal signals such as $x(t) = A\cos(\omega t + \phi)$ as complex exponentials $z(t) = Ae^{j\phi}e^{j\omega t}$. The key is to use the complex amplitude and then the real part operator applied to Euler's formula:

$$x(t) = A\cos(\omega t + \phi) = \Re e\{Ae^{j\phi}e^{j\omega t}\}$$

Manipulating sinusoidal functions using complex exponentials turns trigonometric problems into simple arithmetic and algebra. In this lab, we first review the complex exponential signal and the phasor addition property needed for adding cosine waves. Then we will use MATLAB to make plots of phasor diagrams that show the vector addition needed when combining sinusoids.

1.1 Complex Numbers in MATLAB

MATLAB can be used to compute complex-valued formulas and also to display the results as vector or "phasor" diagrams. For this purpose several new MATLAB functions have been written and are available on the *SP First CD-ROM*. Make sure that this toolbox has been installed[1] by doing `help` on the new M-files: `zvect`, `zcat`, `ucplot`, `zcoords`, and `zprint`. Each of these functions can plot (or print) several complex numbers at once, when the input is formed into a vector of complex numbers. For example, try the following function call and observe that it will plot five vectors all on one graph:

CD-ROM

SP First MATLAB Toolbox

```
zvect( [ 1+j, j, 3-4*j, exp(j*pi), exp(2j*pi/3) ] )
```

Here are some of MATLAB's complex number operators:

`conj`	Complex conjugate
`abs`	Magnitude
`angle`	Angle (or phase) in radians
`real`	Real part
`imag`	Imaginary part
`i,j`	pre-defined as $\sqrt{-1}$
`x = 3 + 4i`	`i` suffix defines imaginary constant (same for `j` suffix)
`exp(j*theta)`	Function for the complex exponential $e^{j\theta}$

Each of these functions takes a vector (or matrix) as its input argument and operates on each element of the vector. Notice that the function names `mag()` and `phase()` do not exist in MATLAB.[2]

Finally, there is a complex numbers drill program called:

CD-ROM

Z Drill

```
zdrill
```

which uses a GUI to generate complex number problems and check your answers. *Please spend some time with this drill since it is very useful in helping you to get a feel for complex arithmetic.*

When unsure about a command, use `help`.

1.2 Sinusoid Addition Using Complex Exponentials

Recall that sinusoids may be expressed as the real part of a complex exponential:

$$x(t) = A\cos(2\pi f_0 t + \phi) = \Re e\left\{Ae^{j\phi}e^{j2\pi f_0 t}\right\} \tag{1}$$

The *Phasor Addition Rule* presented in Section 2.6.2 of the text shows how to add several sinusoids:

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_0 t + \phi_k) \tag{2}$$

[1]Correct installation means that the `spfirst` directory will be on the MATLAB path. Try `help path` if you need more information.

[2]In the latest release of MATLAB a function called `phase()` is defined in a seldom used toolbox; it does more or less the same thing as `angle()` but also attempts to add multiples of 2π when processing a vector.

assuming that each sinusoid in the sum has the *same* frequency, f_0. This sum is difficult to simplify using trigonometric identities, but it reduces to an algebraic sum of complex numbers when solved using complex exponentials. If we represent each sinusoid with its *complex amplitude*

$$X_k = A_k e^{j\phi_k} \tag{3}$$

Then the complex amplitude of the sum is

$$X_s = \sum_{k=1}^{N} X_k = A_s e^{j\phi_s} \tag{4}$$

Based on this complex number manipulation, the *Phasor Addition Rule* implies that the amplitude and phase of $x(t)$ in equation (2) are A_s and ϕ_s, so

$$x(t) = A_s \cos(2\pi f_0 t + \phi_s) \tag{5}$$

We see that the sum signal $x(t)$ in (2) and (5) is a single sinusoid that still has the same frequency, f_0, and it is periodic with period $T_0 = 1/f_0$.

1.3 Harmonic Sinusoids

There is an important extension where $x(t)$ is the sum of N cosine waves whose frequencies (f_k) are *different*. If we concentrate on the case where the frequencies (f_k) are all multiples of one basic frequency f_0, i.e.,

$$f_k = k f_0 \qquad \text{(HARMONIC FREQUENCIES)}$$

then the sum of N cosine waves given by (2) becomes

$$x_h(t) = \sum_{k=1}^{N} A_k \cos(2\pi k f_0 t + \phi_k) = \Re e\left\{\sum_{k=1}^{N} X_k\, e^{j2\pi k f_0 t}\right\} \tag{6}$$

This particular signal $x_h(t)$ has the property that it is also periodic with period $T_0 = 1/f_0$, because each of the cosines in the sum repeats with period T_0. The frequency f_0 is called the *fundamental frequency*, and T_0 is called the *fundamental period*. (Unlike the single frequency case, there is no phasor addition theorem here to combine the harmonic sinusoids.)

2 Pre-Lab

You need to do all exercises in this section to be able to solve the on-line pre-lab exercise.

2.1 Complex Numbers

This section will test your understanding of complex numbers when plotted as vectors. Use $z_1 = 2e^{j\pi/4}$ and $z_2 = -\sqrt{3} + j$ for all parts of this section.

(a) Enter the complex numbers z_1 and z_2 in MATLAB and plot them with `zvect()`, and print them with `zprint()`.

When unsure about a command, use `help`.

Whenever you make a plot with `zvect()` or `zcat()`, it is helpful to provide axes for reference. An x-y axis and the unit circle can be superimposed on your `zvect()` plot by doing the following:

```
hold on, zcoords, ucplot, hold off
```

(b) Compute the conjugate z^* and the inverse $1/z$ for both z_1 and z_2 and plot the results. In MATLAB, see `help conj`. Display the results numerically with `zprint`.

(c) The function `zcat()` can be used to plot vectors in a "head-to-tail" format. Execute the statement `zcat([1+j,-2+j,1-2j]);` to see how `zcat()` works when its input is a vector of complex numbers.

(d) Compute $z_1 + z_2$ and plot the sum using `zvect()`. Then use `zcat()` to plot z_1 and z_2 as 2 vectors head-to-tail, thus illustrating the vector sum. Use `hold on` to put all 3 vectors on the same plot. If you want to see the numerical value of the sum, use `zprint()` to display it.

(e) Compute $z_1 z_2$ and z_2/z_1 and plot the answers using `zvect()` to show how the angles of z_1 and z_2 determine the angles of the product and quotient. Use `zprint()` to display the results numerically.

(f) Make a 2×2 subplot that displays four plots in one window: similar to the four operations done previously: (i) z_1, z_2, and the sum $z_1 + z_2$ on a single plot; (ii) z_2 and z_2^* on the same plot; (iii) z_1 and $1/z_1$ on the same plot; and (iv) $z_1 z_2$. Add a unit circle and x-y axis to each plot for reference.

2.2 Z-Drill

Work a few problems generated by the complex number drill program. To start the program simply type `zdrill` (if necessary, install the GUI and add `zdrill` to MATLAB's path). Use the buttons on the graphical user interface (GUI) to produce different problems.

2.3 Vectorization

The power of MATLAB comes from its matrix-vector syntax. In most cases, loops can be replaced with vector operations because functions such as `exp()` and `cos()` are defined for vector inputs, e.g.,

```
cos(vv) = [cos(vv(1)), cos(vv(2)), cos(vv(3)), ...  cos(vv(N))]
```

where `vv` is an N-element row vector. Vectorization can be used to simplify your code. If you have the following code that plots a certain signal,

```
M = 200;
for k=1:M
   x(k) = k;
   y(k) = cos( 0.001*pi*x(k)*x(k) );
end
plot( x, y, 'ro-' )
```

then you can replace the `for` loop and get the same result with 3 lines of code:

```
M = 200;
y = cos( 0.001*pi*(1:M).*(1:M) );
plot( 1:M, y, 'ro-' )
```

Use this vectorization idea to write 2 or 3 lines of code that will perform the same task as the following MATLAB script without using a `for` loop. (Note: there is a difference between the two operations `xx*xx` and `xx.*xx` when `xx` is a vector.)

```
%--- make a plot of a weird signal
N = 200;
for k=1:N
```

```
      xk(k) = k/60;
      rk(k) = sqrt( xk(k)*xk(k) - 1 );
      sig(k) = exp(j*2*pi*rk(k));
   end
   plot( xk, real(sig), mo-, xk, imag(sig), go- )
```

2.4 Functions

Functions are a special type of M-file that can accept inputs (matrices and vectors) and also return outputs. The keyword `function` must appear as the first word in the ASCII file that defines the function, and the first line of the M-file defines how the function will pass input and output arguments. The file extension must be lower case "m" as in `my_func.m.` See Section B-6 in Appendix B of the text for more discussion.

The following function has few mistakes. Before looking at the correct one below, try to find these mistake(s) (there are at least three):

```
matlab mfile  [xx,tt] = badcos(ff,dur)
%BADCOS  Function to generate a cosine wave
%  usage:
%       xx = badcos(ff,dur)
%       ff = desired frequency
%      dur = duration of the waveform in seconds
%
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
badcos = cos(2*pi*freeq*tt);
```

The corrected function should look something like:

```
function [xx,tt] = goodcos(ff,dur)
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
xx = cos(2*pi*ff*tt);
```

Notice the word "function" in the first line. Also, "freeq" has not been defined before being used. Finally, the function has "xx" as an output and hence "xx" should appear in the left-hand side of at least one assignment line within the function body. The function name is *not* used to hold values produced in the function.

3 Warm-Up: Complex Exponentials

In the Pre-Lab part of this lab, you learned how to write function M-files. In this section, you will write two functions that can generate sinusoids, or sums of sinusoids.

3.1 Vectorization

```
%--- make a plot of a 2-D function
x = 1:50;   y = -30:30;
F = zeros(length(x),length(y));
for kx = 1:length(x)
    for ky = 1:length(y)
        theta = atan2(y(ky),x(kx));
        rr = abs(x(kx)+j*y(ky));
        F(kx,ky) = rr*cos(6*theta);
    end
end
imagesc(x,y,rot90(F))  %- Rotate matrix to make columns of F be the y-axis
grid on, axis xy, shg, colorbar
```

Use the vectorization idea to write 1 or 2 lines of code that will perform the same task as inner loop of the preceding MATLAB script without using a `for` loop. If you are ambitious, look up the MATLAB function `meshgrid` and use it to replace both loops with some vectorized code.
Note: there is a difference between the two operations `rr*rr` and `rr.*rr` when `rr` is a vector.

Instructor Verification (separate page)

3.2 M-file to Generate a Sinusoid

Write a function that will generate a single sinusoid, $x(t) = A\cos(\omega t + \phi)$, by using five input arguments: amplitude (A), phase (ϕ), frequency (ω), starting time (`tstart`), and duration (`dur`). The function should return two outputs: the values of the sinusoidal signal (x) and corresponding times (t) at which the sinusoid values are known. Make sure that the function generates exactly 40 values of the sinusoid per period. Call this function `one_cos()`. *Hint: use* `goodcos()` *from the Pre-Lab part as a starting point.* Plot the result from the following call to test your function.

```
[xx0,tt0] = one_cos( [5], [-pi/3], [2], -1, 2);
```

3.3 Sinusoidal Synthesis with an M-file: Different Frequencies

Since we will generate many functions that are a "sum of sinusoids," it will be convenient to have a function for this operation. To be general, we will allow the frequency of each component (f_k) to be different. The following expressions are equivalent if we define the complex amplitude X_k as $X_k = A_k e^{j\phi_k}$.

$$x(t) = \Re e\left\{\sum_{k=1}^{N}(A_k e^{j\phi_k})e^{j2\pi f_k t}\right\} \tag{7}$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) \tag{8}$$

3.3.1 Write the Function M-file

Write an M-file called `makecos.m` that will synthesize a waveform in the form of (7). Although `for` loops are rather inefficient in MATLAB, *you must write the function with one outer loop in this lab.* The inner loop can be vectorized. The first few statements of the M-file are the comment lines—they should look like:

```
function    [xx,tt] = makecos(Ak, phik, fk, tstart, dur)
%MAKECOS  Function to synthesize a sum of cosine waves
%  usage:
%    [xx,tt] = makecos(Ak, phik, fk, tstart, dur)
%      Ak = vector of amplitudes
%    phik = vector of phases
%      fk = vector of frequencies   (all positive)
%  tstart = starting time
%     dur = total time duration of the signal
%      xx = vector of sinusoidal values
%      tt = vector of times, for the time axis
%
%    Note: fk, Ak, and phik must all be the same length.
%             Ak(1) and phik(1) corresponds to frequency fk(1),
%             Ak(2) and phik(2) corresponds to frequency fk(2),
%    The tt vector should be generated with a small time increment that
%         creates 40 samples per period. Use the period corresponding to
%         the highest frequency in the fk vector.
```

The MATLAB syntax `length(fk)` returns the number of elements in the vector `fk`, so we do not need a separate input argument for the number of frequencies. On the other hand, the programmer (that's you) should provide error checking to make sure that the lengths of `fk`, `Ak` and `phik` are all the same. See `help error`.

3.3.2 Testing

In order to verify that this M-file can synthesize sinusoids, try the following test and plot the result.

```
[xx0,tt0] = makecos( [5,2], [-pi/3,pi], [2,0], -1, 2);
```

Measure the DC value and the period of `xx0` by hand. Then write an explanation on the verification sheet of why the measured values are correct.

Instructor Verification (separate page)

4 Lab Exercises: Direction Finding

Why do humans have two ears? One answer is that the brain can process acoustic signals received at the two ears and determine the direction to the source of the acoustic energy. Using sinusoids, we can describe and analyze a simple scenario that demonstrates "direction finding" in terms of phase differences. This same principle is used in many other applications including radars that locate and track airplanes.

4.1 Direction Sensitivity with Microphones

Consider a simple measurement system that consists of two microphones that can both hear the same source signal. If the microphones are placed some distance apart, then the sound must travel different paths from the source to the receivers. When the travel paths have different lengths, the two signals will arrive at different times.

The received signal at a receiver, called $r(t)$, is a delayed copy of the transmitter signal $s(t)$. Thus we could write

$$r(t) = s(t - t_d)$$

where t_d is the amount of time it takes for the signal to travel from the source to the receiver and $s(\cdot)$ is the transmitted (sinusoidal) signal.[3] The travel time t_d can be computed easily once we know the speed of sound and the locations of the source and receiver.

If we assume that the source signal is a sinusoid, we can combine the signals from two receivers in many different ways. Consider the case of one source transmitting the signal $s(t)$ to two receivers. The received signals could be label as

$$\begin{aligned} \text{Receiver \#1:} \quad & r_1(t) = s(t - t_1) \\ \text{Receiver \#2:} \quad & r_2(t) = s(t - t_2) \end{aligned}$$

where t_1 is the propagation time from the source to Receiver #1, and t_2 the propagation time from the source to Receiver #2. One simple method is to *subtract* the two received signals.

$$y(t) = r_1(t) - r_2(t)$$

In this case, the output of the subtraction would be zero when the two signals are identical, or, equivalently, when $t_1 = t_2$. Figure 1 shows the two-receiver system with a transmitter position that will produce zero output.

The goal in this part of the lab will be to produce a plot of the "receiver differencing output" as a function of the (x, y) location of the transmitter over a region as shown in Fig. 2.

(a) The amount of the delay (in seconds) can be computed for both propagation paths. First of all, consider the path from the transmitter to Receiver #1. The time delay is the distance from the transmitter location (x_t, y_t) to the receiver at $(0, y_1)$, divided by the speed of sound which is approximately $c = 330$ m/s. Write a mathematical expression for this time delay, which will be called t_1.

[3]For simplicity we will ignore propagation losses. Usually, the amplitude of an acoustic signal that propagates over a distance R will be reduced by an amount that is inversely proportional to R.

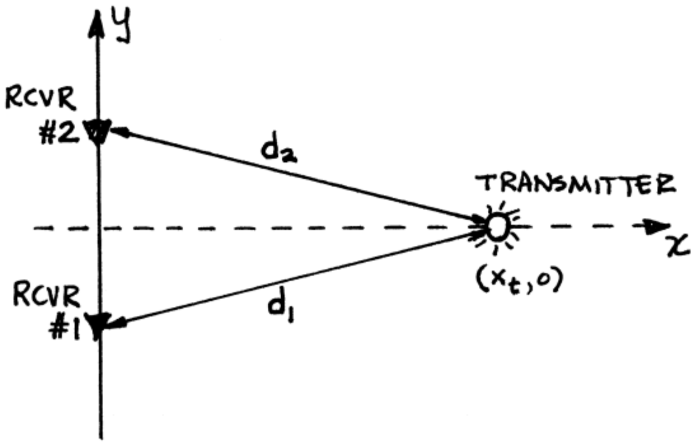

Figure 1: Scenario for two-receiver system. If the receiver signals are subtracted, then the output will be zero whenever the transmitter is located on the centerline that bisects the line segment between the two receivers.

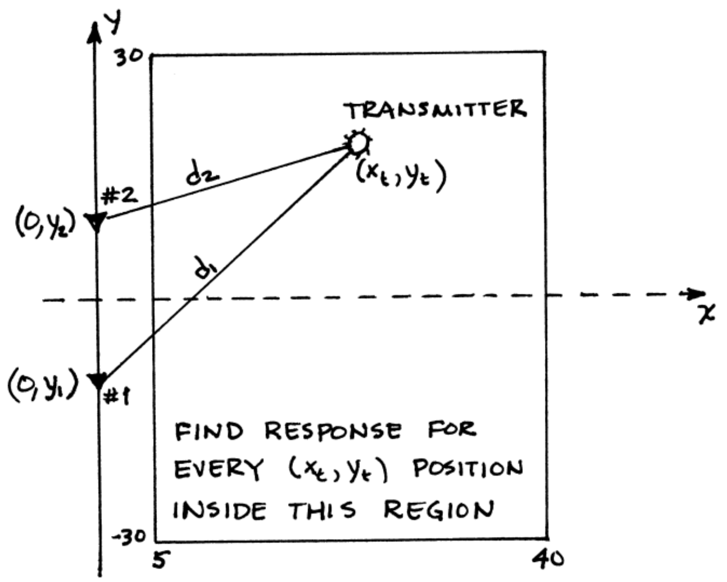

Figure 2: Region where the response should be calculated.

(b) Now write a mathematical formula for the time delay of the signal that travels the path from the transmitter at (x_t, y_t) to Receiver #2 at $(0, y_2)$. Call this delay time t_2 and write a mathematical expression for t_2 in terms of d_2, the distance from transmitter to receiver.

(c) The received signals at the receivers, called $r_1(t)$ and $r_2(t)$, are delayed copies of the transmitter signal. Assume that the source signal $s(t)$ is a sinusoid with a frequency of $f_0 = 60$ Hz; and also assume that the complex amplitude of the transmitted signal is $10e^{-j\pi/3}$. Make a plot of $r_1(t)$ and $r_2(t)$ when $x_t = 50$ meters, and $y_t = 0$. Assume that the receivers are located at $y_1 = -8$ meters, and $y_2 = 8$ meters. Use `subplot` to put both signals on the same figure. Plot only 3 periods of the received sinusoids and then measure the *relative time-shift* between the two received signals by comparing peak locations. Explain why the relative time-shift turns out to be zero.

(d) Repeat the operations of the previous part for a different transmitter location: $x_t = 25$ meters, and $y_t = -10$ meters. Assume the same transmitter frequency and the same receiver locations. Measure the peak amplitude of the output sinusoid that you will have in $y(t)$.

(e) Describe how the calculation of peak amplitude in the previous part could have been done with a single phasor subtraction.

(f) The objective in the rest of this lab is to write a MATLAB function that will determine the peak amplitude of the output for all possible transmitter locations. To do this, the received signals should be manipulated as complex amplitudes, not as sinusoidal waveforms.

The MATLAB function that you write should be similar to the vectorization example in Section 3.1 in the warmup. The result of your function should be a 2-D image plot that shows the output amplitude for (x, y) in the region:

$$5 \leq x \leq 40 \qquad \text{and} \qquad -30 \leq y \leq 30$$

The complex amplitudes at the receivers $R_1 = A_1 e^{j\phi_1}$ and $R_2 = A_2 e^{j\phi_2}$ can be found by calculating the distances from the transmitter to each receiver, and using the speed of sound (c) to convert to time delay.

(g) *Vectorization:* It is likely that your previous programming skills would lead you to write a double loop to do this implementation. The loop would run over all possible locations of the transmitter (x_t, y_t), and would do the subtraction of the two receivers for each transmitter position, one at a time.
However, it is possible to vectorize this program similar to what was done for the example in Section 3.1. Replacing one loop with a vectorized statement should not be too hard. If you like a challenge, try to eliminate both loops.
Hint: the MATLAB function `meshgrid` would provide a way to generate all the (x_t, y_t) positions at once.

(h) Use the 2-D image plot of the output amplitude to show that this simple "differencing processor" will have zero output at locations other than along the centerline (as in Fig. 1). In fact, if we define the centerline to be at an angle of zero degrees measured with respect to the x-axis, determine other angles where the output is also zero.

4.1.1 Miscellaneous Comments

In order to generate a good 2-D image plot of the response it will be necessary to have a dense grid over the region of interest. A dense grid will make it easy to find the other locations where the response is zero. However, as the grid density goes up, the running time of the program will also increase. Doubling the grid density in both x and y would quadruple the running time.

The are numerous ways to plot a 2-D function of (x, y): `imagesc`, `contour` or `mesh`. In this lab, the `contour` function might make it easier to answer the question about angles.

Why would we be interested in the other places where the response is zero? Suppose that the two receivers were being used to point in the direction of the transmitter, under the assumption that zero response happens along the centerline between the receivers. If there are other places where the output is zero, then the pointing would be ambiguous because we could not guarantee that zero output corresponds to one direction.

Lab #2
ECE-2025
Spring-2003
INSTRUCTOR VERIFICATION SHEET

Turn this page in to your TA before the end of your lab period.

Name: ______________________________ Date of Lab: __________

Part 3.1 Replace the inner `for` loop with only 1 or 2 lines of vectorized MATLAB code. Write the MATLAB code in the space below:

Verified:____________________ Date/Time:__________

Part 3.3.2 Show that your `makecos.m` function is correct by running the test in Section 3.3.2 and plotting the result. Measure the DC value and the period of `xx0` and explain why the measured values are correct. Write your explanations in the space below.

Verified:____________________ Date/Time:__________