

1 Spherical Cap

1.a Area and Volume

In this part all you have to do is to transfer the given mathematical formulas into assignments using arithmetic operators in MATLAB. When you run the script file, *test_spherical_cap.m*, it will ask you to enter radius and height, then it will compute area and volume using your formulas, and display the results.

```
function [A,V] = spherical_cap(r,h)
% Returns the area and volume of a spherical cap
% cut from a sphere of radius r, and cap height h

%=====

A = 2*pi*r*h;
V = pi*h^2*(3*r-h)/3;

%=====
```

1.b Base Radius

The second part asks you to expand the list of outputs to include the base radius. Using Pythagorean theorem $r^2 = a^2 + (r - h)^2$, we can compute **a**. Notice the additional output **a** in the first line, and the corresponding assignment in the last line.

```
function [A,V,a] = spherical_cap(r,h)
% Returns the area, volume and radius of a spherical cap
% cut from a sphere of radius r, and cap height h

%=====

A = 2*pi*r*h;
V = pi*h^2*(3*r-h)/3;
a = sqrt(r^2 - (r-h)^2);

%=====
```

If you wanted to test the new functionality, you could make the following changes in *test_spherical_cap.m*. The line numbers are denoted on the left.

```
22 [area, volume, base-radius] = spherical_cap(radius,height);
```

```
36 fprintf('Computed Base Radius: %f \n', base.radius);
```

For sanity checks, you could use *zero cap* $r = 1, h = 0$; a *hemisphere* $r = 1, h = 1$ and something like $r = 5, h = 2$.

2 Triangle

2.a Coordinates

In this part you should use `input` function in MATLAB to ask for the 6 coordinates, and store them in variables.

```
% Ask the user for x,y coordinates of 3 points
x1 = input('Enter x-coordinate for point 1: ');
y1 = input('Enter y-coordinate for point 1: ');

x2 = input('Enter x-coordinate for point 2: ');
y2 = input('Enter y-coordinate for point 2: ');

x3 = input('Enter x-coordinate for point 3: ');
y3 = input('Enter y-coordinate for point 3: ');
```

2.b Shoelace Formula

You should use the given formula to compute the area. For absolute value we use `abs` function. It is more appropriate to use a variable name such as `area.shoelace`, `AS` etc. rather than `A` or `A1`.

```
% Compute the area using shoelace formula and display the result
shoelace = 0.5 * abs(x1*y2 + x2*y3 + x3*y1 - x2*y1 - x3*y2 - x1*y3);
fprintf('Area using shoelace formula : %.15f \n', shoelace);
```

2.c Heron's Formula

Again we will use the formulas provided. Notice that we need to do more computations in this part.

```
% Compute the side lengths
a = sqrt((x1-x2)^2+(y1-y2)^2);
b = sqrt((x2-x3)^2+(y2-y3)^2);
c = sqrt((x3-x1)^2+(y3-y1)^2);

% Compute semiperimeter
s = (a+b+c)/2;

% Compute the area using Heron's formula and display the result
heron = sqrt(s*(s-a)*(s-b)*(s-c));
fprintf('Area using Heron's formula : %.15f \n', heron);
```

2.d Equal?

```
% Check if the computed results are the same
if shoelace == heron
    disp('the areas are equal')
else
    disp('the areas are not equal')
end
```

What you will notice is that for many of the triangles the code will display that the results are not equal! If you check the numbers by just removing the semicolon after area assignments, or just using plain `disp` function, or using only `%f` in your `fprintf` strings, then you are going to see the same values displayed even though you get a *not equal* message. Using `format long` you can increase the number of digits displayed after the decimal point. Or you change the formatting in your `fprintf`. For example `%.15f` might reveal the slight difference in your results. Be careful when you compare decimal numbers!

```
% Ask the user for x,y coordinates of 3 points
x1 = input('Enter x-coordinate for point 1: ');
y1 = input('Enter y-coordinate for point 1: ');

x2 = input('Enter x-coordinate for point 2: ');
y2 = input('Enter y-coordinate for point 2: ');

x3 = input('Enter x-coordinate for point 3: ');
y3 = input('Enter y-coordinate for point 3: ');

%format long

% Compute the area using shoelace formula and display the result
shoelace = 0.5 * abs(x1*y2 + x2*y3 + x3*y1 - x2*y1 - x3*y2 - x1*y3);
fprintf('Area using shoelace formula : %.15f \n', shoelace);

% Compute the side lengths
a = sqrt((x1-x2)^2+(y1-y2)^2);
b = sqrt((x2-x3)^2+(y2-y3)^2);
c = sqrt((x3-x1)^2+(y3-y1)^2);

% Compute semiperimeter
s = (a+b+c)/2;

% Compute the area using Heron's formula and display the result
heron = sqrt(s*(s-a)*(s-b)*(s-c));
fprintf('Area using Heron''s formula : %.15f \n', heron);

% Check if the computed results are the same
if shoelace == heron
    disp('the areas are equal')
else
    disp('the areas are not equal')
end
```

3 Die Roll

3.a roll

Do you remember the number guessing game from our second lecture? We generated integers from 0 to 9 using `fix(10*rand)`. Here `fix` function rounds numbers towards zero. You can change the range from $(0,1)$ to $(0,n)$ by changing the multiplier in front of `rand` function. A die has six faces, so `fix(6*rand)+1` is a solution. Here you can also use `floor`. `ceil(6*rand)` is even better. Since in general `roll` function should generate integer values from 1 to n , we should use n instead of 6. If we don't provide any input to this function it will already assume $n = 6$.

```
function result = roll(n)
% Returns an integer from 1 to n with equal probabilities.
% If n is not provided, it assumes n = 6 and emulates a die.

if nargin < 1    % if no input argument is provided
    n = 6;      % make an assumption.
end

assert(n > 1);

%=====

result = ceil(n*rand);

%=====
```

A fair die should have equal probabilities for each face. In *test_roll.m*, when you increase the number of rolls N , you should see the heights of the bars getting closer to each other.

3.b randi

When you ask for help on `rand` function, you get a description and example usage. You also get a list of similar functions at the end of the text. One of those functions `randi` has the same functionality our `roll` function provides (and even more than that). In the file *test_roll.m* we could change the following line, and it would produce similar results.

```
11 data(j) = randi(6); % We can use randi function instead of roll.
```

4 Thick Coin

Imagine the smallest sphere the coin can fit into. When the coin is tossed, based on which point on this imaginary sphere touches the ground first, the result of the toss can be determined. The coin separates the spherical surface into 3 parts: 2 spherical caps and a strip around the equator. The caps correspond to heads and tails, and the strip corresponds to the side, i.e. the base of a spherical cap matches one face of the coin. The probabilities are proportional to these areas covered on the sphere.

If the thickness of the coin is represented with t , then $h = r - t/2$. And the radius of the coin is $a = \sqrt{r^2 - t^2/4}$. We already know how to compute the area corresponding to heads (or tails) $A_H = 2\pi r(r - t/2)$. The area for a hemisphere, when $h = r$, can be calculated as $A = 2\pi r^2$. The area of the strip corresponding to the side is

$$\begin{aligned} A_S &= 2 \times (\text{hemisphere} - \text{cap}) \\ &= 2 \times (2\pi r^2 - 2\pi r(r - t/2)) \\ &= 2\pi r t \end{aligned}$$

We want the probabilities be equal $P_H = P_T = P_S = 1/3$, and thus the areas should satisfy $A_H = A_T = A_S$.

$$2\pi r(r - t/2) = 2\pi r t \Rightarrow r = 3t/2$$

The question asks the value for t/a which can be computed as

$$\begin{aligned} \frac{t}{a} &= \frac{t}{\sqrt{r^2 - t^2/4}} \\ &= \frac{t}{\sqrt{9t^2/4 - t^2/4}} \\ &= \frac{1}{\sqrt{2}} \approx 0.707 \end{aligned}$$

In this geometric model we assumed that the coin can be oriented in any direction. Is it a correct assumption? Check out the course website for your entertainment.

How do we solve this problem with MATLAB? The *wording* of the problem sets a critical equality for all three possible outcomes. However what we might be interested to know can be how the probability of landing on the side changes based on thickness to radius ratio of the coin. We could just write a script to display P_H , P_T and P_S after we enter values for t and a . But you should realize that they depend only on the ratio t/a . Keeping one of them fixed and playing with the other we could seek for the solution. Or simply enter the ratio for t and set $a = 1$.

```
% Ask for coin dimensions
t = input('Enter thickness: ');
a = input('Enter radius: ');

r = sqrt(a^2 + t^2/4); % radius of the imaginary sphere
h = r - t/2;          % height of the spherical caps

% Compute areas
AH = spherical_cap(r,h);
AT = AH;
A = 2*spherical_cap(r,r);
AS = A - AH - AT;

PH = AH/A;
PT = AT/A;
PS = AS/A;

fprintf('Heads: %f \nTails: %f\nSide : %f\n', PH, PT, PS);
```

You could have written a function to compute the probabilities and used it in your script as well:

```
function [PH,PT,PS] = thick_coin_probabilities(t,a)
% This function computes the probabilities for heads, tails and the side
% for a coin with thickness t, and radius a. We assume a geometric model
% with no bouncing.

r = sqrt(a^2+t^2/4);
h = r - t/2;

A = 2*spherical_cap(r,r); % Full sphere or simply 4*pi*r^2
AH = spherical_cap(r,h); % Area of cap corresponding to heads
% AT is equal to AH, we don't need to compute.

PH = AH/A; % Probabilities are proportional to areas
PT = PH; % Since AH = AT
PS = 1-2*PH; % Since PH + PT + PS = 1
```

Usually plots provide much more information and give a sense for values which we can't get directly from formulas unless we are familiar with them. On the following page you can see the probabilities for heads and the side plotted as we change the thickness to radius ratio for the coin. In the question we are only asked for the indicated intersection point, where all probabilities are equal to $1/3$. Yet we have a better understanding about the dependence on thickness when we see a plot. Also note that for zero thickness $P_H = 1/2$ (i.e. an idealized coin) and for extremely thick coin (it doesn't look like a coin at all) it almost always falls on the side: $P_S \rightarrow 1$ as t/a is too large.

We will cover how to create plots later in the course.

