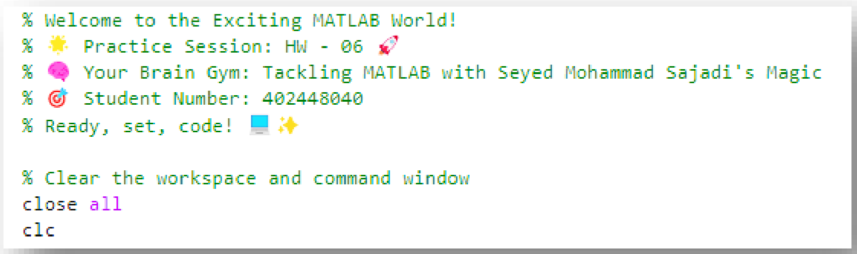
I work with MATLAB 2023b, please be careful.

1. **Header Comments:**

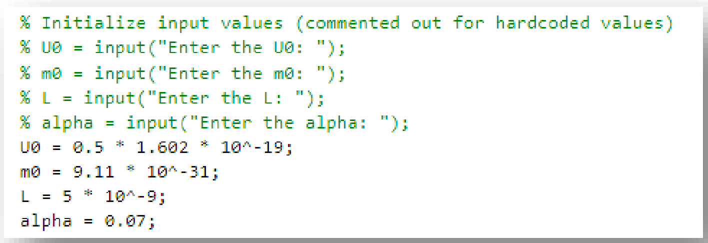
- Provides introductory information about the MATLAB script, including the purpose of the code, the student's details, and a fun welcome message.



```
% Welcome to the Exciting MATLAB World!
% ☀️ Practice Session: HW - 06 🚀
% 🧠 Your Brain Gym: Tackling MATLAB with Seyed Mohammad Sajadi's Magic
% 🎯 Student Number: 402448040
% Ready, set, code! 💻✨

% Clear the workspace and command window
close all
clc
```

2. **Input Initialization:**

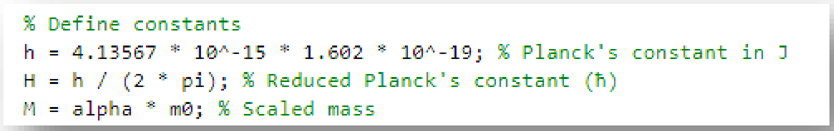
- Initializes input values for physical parameters or uses hardcoded values.
- Sets values for energy, mass, length, and alpha.



```
% Initialize input values (commented out for hardcoded values)
% U0 = input("Enter the U0: ");
% m0 = input("Enter the m0: ");
% L = input("Enter the L: ");
% alpha = input("Enter the alpha: ");
U0 = 0.5 * 1.602 * 10^-19;
m0 = 9.11 * 10^-31;
L = 5 * 10^-9;
alpha = 0.07;
```

3. **Constants Definition:**

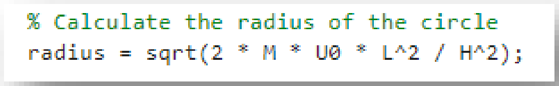
- Defines fundamental constants like Planck's constant, reduced Planck's constant, and scaled mass based on the given parameters.



```
% Define constants
h = 4.13567 * 10^-15 * 1.602 * 10^-19; % Planck's constant in J
H = h / (2 * pi); % Reduced Planck's constant (ħ)
M = alpha * m0; % Scaled mass
```

4. **Circle Radius Calculation:**

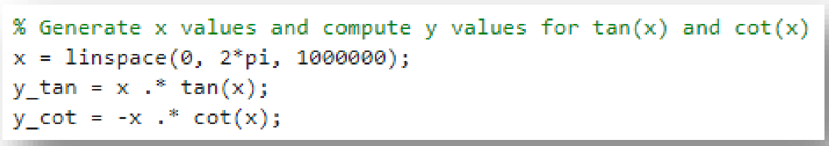
- Uses the defined constants to calculate the radius of a circle in a potential energy landscape.



```
% Calculate the radius of the circle
radius = sqrt(2 * M * U0 * L^2 / H^2);
```

5. **Function Plotting:**

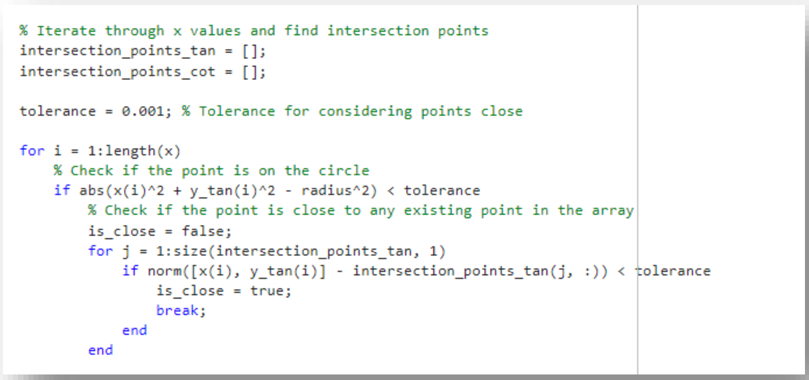
- Generates x values and calculates y values for tangent and cotangent functions.
- Plots these functions along with a circle using parametric equations.



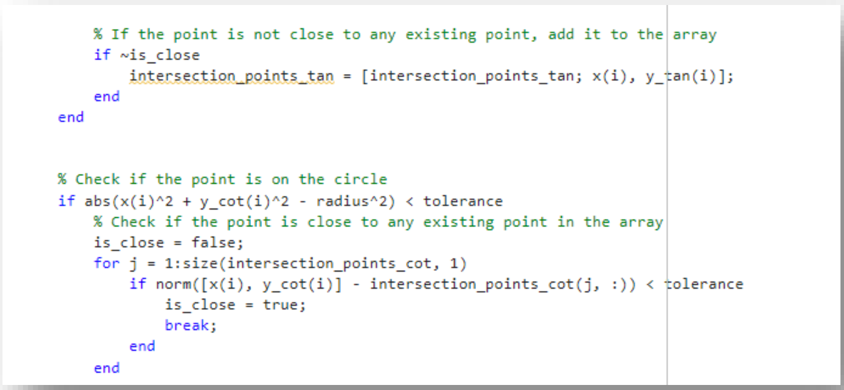
```
% Generate x values and compute y values for tan(x) and cot(x)
x = linspace(0, 2*pi, 1000000);
y_tan = x .* tan(x);
y_cot = -x .* cot(x);
```

6. **Intersection Points Calculation:**

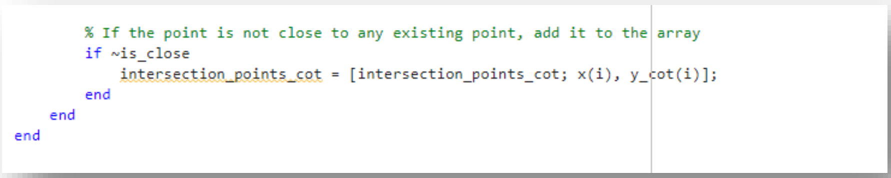
- Iterates through x values to find intersection points of tangent and cotangent functions with the circle.



```
% Iterate through x values and find intersection points
intersection_points_tan = [];
intersection_points_cot = [];

tolerance = 0.001; % Tolerance for considering points close

for i = 1:length(x)
    % Check if the point is on the circle
    if abs(x(i)^2 + y_tan(i)^2 - radius^2) < tolerance
        % Check if the point is close to any existing point in the array
        is_close = false;
        for j = 1:size(intersection_points_tan, 1)
            if norm([x(i), y_tan(i)] - intersection_points_tan(j, :)) < tolerance
                is_close = true;
                break;
            end
        end
```



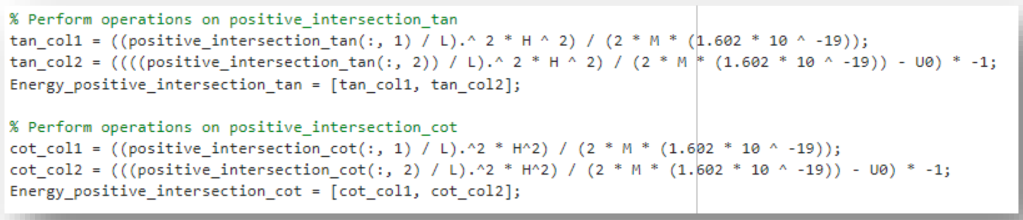
```
        % If the point is not close to any existing point, add it to the array
        if ~is_close
            intersection_points_tan = [intersection_points_tan; x(i), y_tan(i)];
        end
    end


    % Check if the point is on the circle
    if abs(x(i)^2 + y_cot(i)^2 - radius^2) < tolerance
        % Check if the point is close to any existing point in the array
        is_close = false;
        for j = 1:size(intersection_points_cot, 1)
            if norm([x(i), y_cot(i)] - intersection_points_cot(j, :)) < tolerance
                is_close = true;
                break;
            end
        end
```



```
        % If the point is not close to any existing point, add it to the array
        if ~is_close
            intersection_points_cot = [intersection_points_cot; x(i), y_cot(i)];
        end
    end
end
```

7. Energy Calculation:

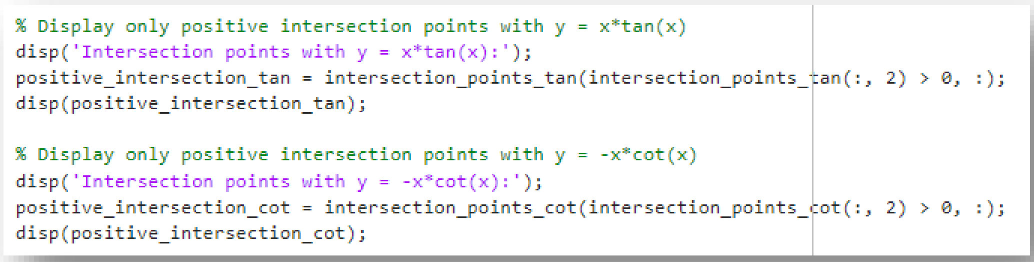
- Calculates energy at the intersection points for both tangent and cotangent functions.



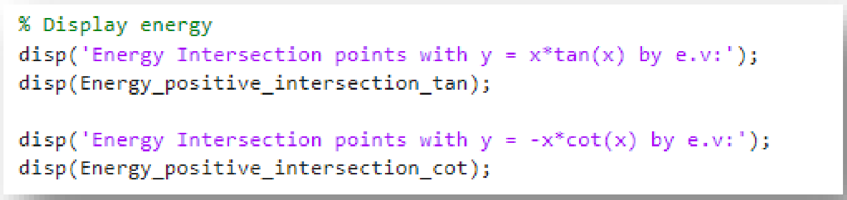
```
% Perform operations on positive_intersection_tan
tan_col1 = ((positive_intersection_tan(:, 1) / L).^ 2 * H ^ 2) / (2 * M * (1.602 * 10 ^ -19));
tan_col2 = ((((positive_intersection_tan(:, 2)) / L).^ 2 * H ^ 2) / (2 * M * (1.602 * 10 ^ -19)) - U0) * -1;
Energy_positive_intersection_tan = [tan_col1, tan_col2];

% Perform operations on positive_intersection_cot
cot_col1 = ((positive_intersection_cot(:, 1) / L).^2 * H^2) / (2 * M * (1.602 * 10 ^ -19));
cot_col2 = (((positive_intersection_cot(:, 2) / L).^2 * H^2) / (2 * M * (1.602 * 10 ^ -19)) - U0) * -1;
Energy_positive_intersection_cot = [cot_col1, cot_col2];
```

8. Result Display:

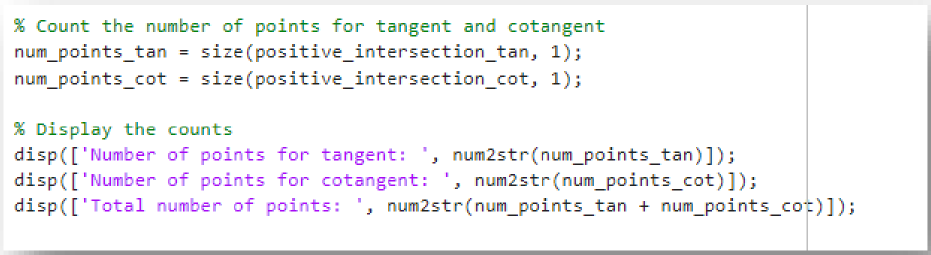
- Displays the intersection points and corresponding energies for both functions.
- Counts and displays the number of points for tangent and cotangent.



```
% Display only positive intersection points with y = x*tan(x)
disp('Intersection points with y = x*tan(x):');
positive_intersection_tan = intersection_points_tan(intersection_points_tan(:, 2) > 0, :);
disp(positive_intersection_tan);

% Display only positive intersection points with y = -x*cot(x)
disp('Intersection points with y = -x*cot(x):');
positive_intersection_cot = intersection_points_cot(intersection_points_cot(:, 2) > 0, :);
disp(positive_intersection_cot);
```



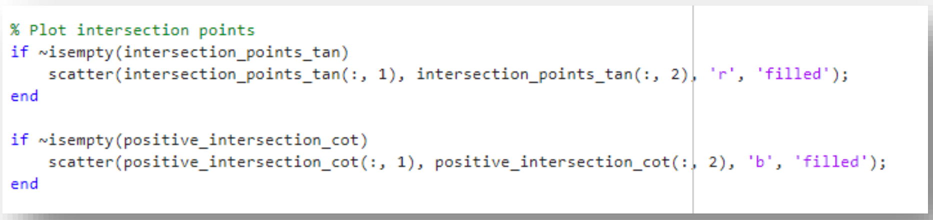
```
% Display energy
disp('Energy Intersection points with y = x*tan(x) by e.v:');
disp(Energy_positive_intersection_tan);

disp('Energy Intersection points with y = -x*cot(x) by e.v:');
disp(Energy_positive_intersection_cot);
```



```
% Count the number of points for tangent and cotangent
num_points_tan = size(positive_intersection_tan, 1);
num_points_cot = size(positive_intersection_cot, 1);

% Display the counts
disp(['Number of points for tangent: ', num2str(num_points_tan)]);
disp(['Number of points for cotangent: ', num2str(num_points_cot)]);
disp(['Total number of points: ', num2str(num_points_tan + num_points_cot)]);
```

9. **Intersection Points Plotting:**

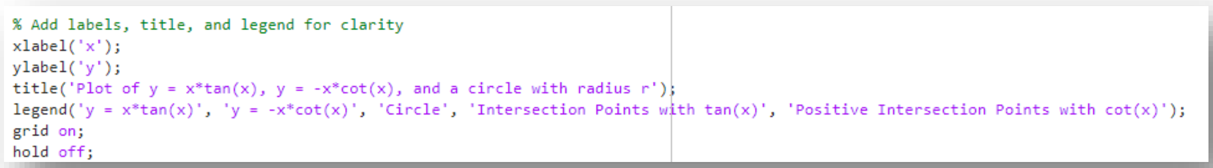
- Plots intersection points on the tangent and cotangent functions.



```
% Plot intersection points
if ~isempty(intersection_points_tan)
    scatter(intersection_points_tan(:, 1), intersection_points_tan(:, 2), 'r', 'filled');
end

if ~isempty(positive_intersection_cot)
    scatter(positive_intersection_cot(:, 1), positive_intersection_cot(:, 2), 'b', 'filled');
end
```

10. **Additional Plotting:**

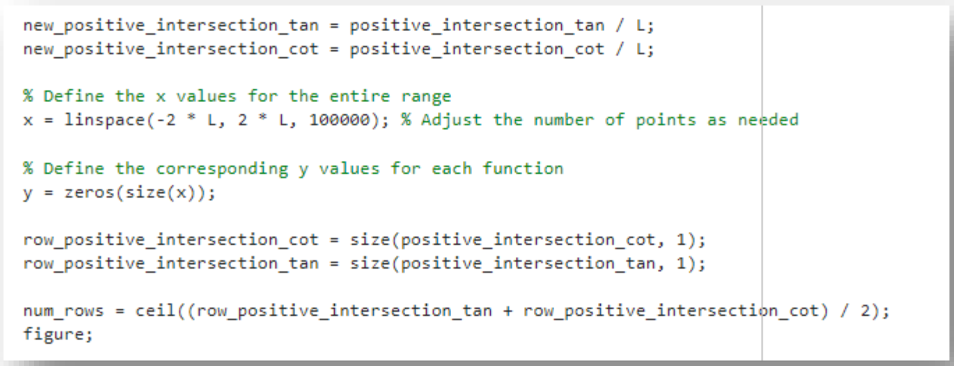
- Creates new variables for scaled intersection points.
- Plots wave functions in subplots based on the calculated intersection points.



```
% Add labels, title, and legend for clarity
xlabel('x');
ylabel('y');
title('Plot of y = x*tan(x), y = -x*cot(x), and a circle with radius r');
legend('y = x*tan(x)', 'y = -x*cot(x)', 'Circle', 'Intersection Points with tan(x)', 'Positive Intersection Points with cot(x)');
grid on;
hold off;
```

11. **Subplot Generation:**

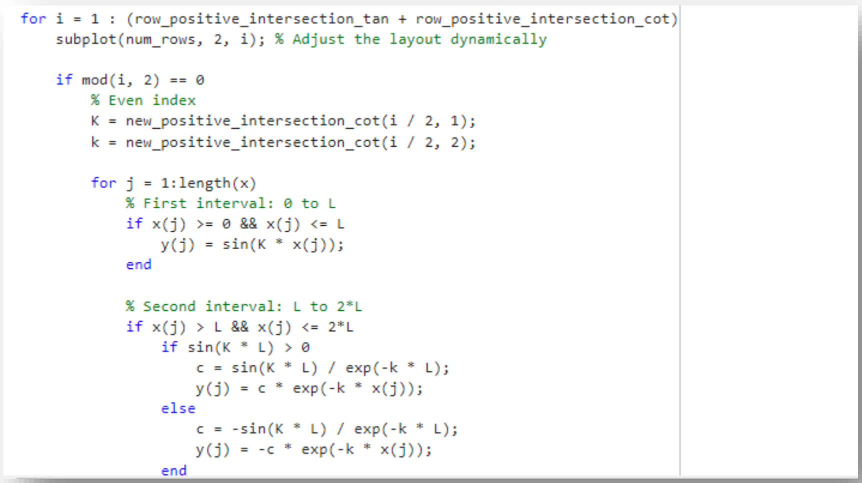
- Iterates through intersection points and plots corresponding wave functions in separate subplots.



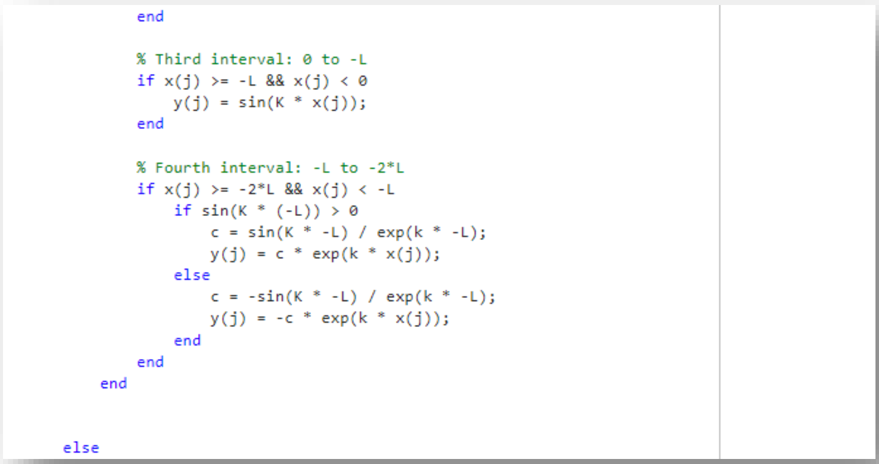
```
new_positive_intersection_tan = positive_intersection_tan / L;
new_positive_intersection_cot = positive_intersection_cot / L;

% Define the x values for the entire range
x = linspace(-2 * L, 2 * L, 100000); % Adjust the number of points as needed

% Define the corresponding y values for each function
y = zeros(size(x));

row_positive_intersection_cot = size(positive_intersection_cot, 1);
row_positive_intersection_tan = size(positive_intersection_tan, 1);

num_rows = ceil((row_positive_intersection_tan + row_positive_intersection_cot) / 2);
figure;
```

12. **Closing Comments:**

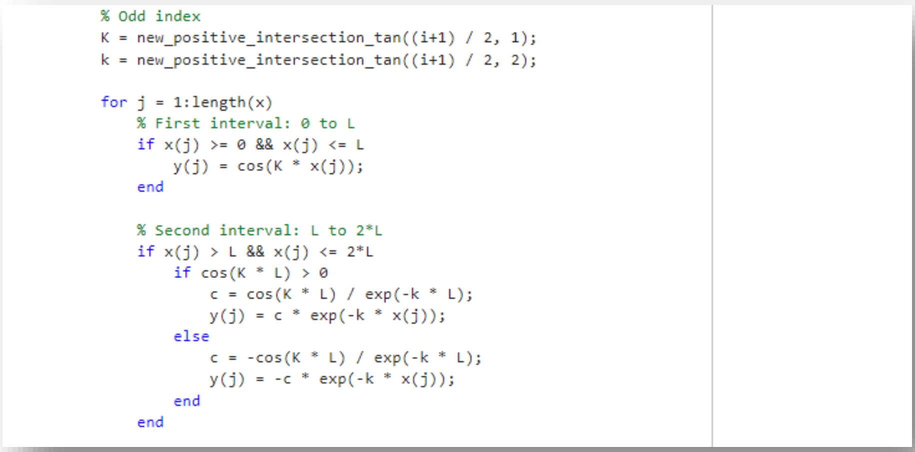
- Ends the script with a comment and ensures clarity in the code structure.



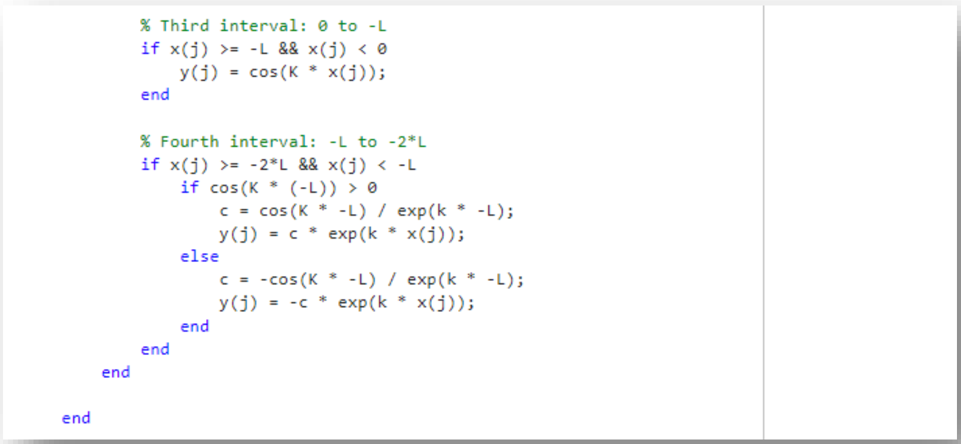
```
for i = 1 : (row_positive_intersection_tan + row_positive_intersection_cot)
    subplot(num_rows, 2, i); % Adjust the layout dynamically

    if mod(i, 2) == 0
        % Even index
        K = new_positive_intersection_cot(i / 2, 1);
        k = new_positive_intersection_cot(i / 2, 2);

        for j = 1:length(x)
            % First interval: 0 to L
            if x(j) >= 0 && x(j) <= L
                y(j) = sin(K * x(j));
            end

            % Second interval: L to 2*L
            if x(j) > L && x(j) <= 2*L
                if sin(K * L) > 0
                    c = sin(K * L) / exp(-k * L);
                    y(j) = c * exp(-k * x(j));
                else
                    c = -sin(K * L) / exp(-k * L);
                    y(j) = -c * exp(-k * x(j));
                end
```



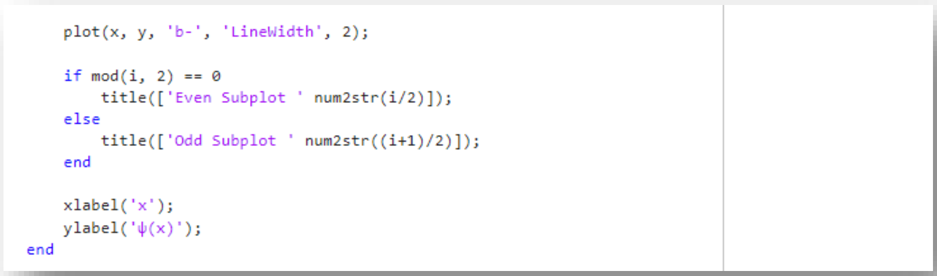
```
            end

            % Third interval: 0 to -L
            if x(j) >= -L && x(j) < 0
                y(j) = sin(K * x(j));
            end

            % Fourth interval: -L to -2*L
            if x(j) >= -2*L && x(j) < -L
                if sin(K * (-L)) > 0
                    c = sin(K * -L) / exp(k * -L);
                    y(j) = c * exp(k * x(j));
                else
                    c = -sin(K * -L) / exp(k * -L);
                    y(j) = -c * exp(k * x(j));
                end
            end
        end


    else
```



```
        % Odd index
        K = new_positive_intersection_tan((i+1) / 2, 1);
        k = new_positive_intersection_tan((i+1) / 2, 2);

        for j = 1:length(x)
            % First interval: 0 to L
            if x(j) >= 0 && x(j) <= L
                y(j) = cos(K * x(j));
            end

            % Second interval: L to 2*L
            if x(j) > L && x(j) <= 2*L
                if cos(K * L) > 0
                    c = cos(K * L) / exp(-k * L);
                    y(j) = c * exp(-k * x(j));
                else
                    c = -cos(K * L) / exp(-k * L);
                    y(j) = -c * exp(-k * x(j));
                end
            end
```



```
            % Third interval: 0 to -L
            if x(j) >= -L && x(j) < 0
                y(j) = cos(K * x(j));
            end

            % Fourth interval: -L to -2*L
            if x(j) >= -2*L && x(j) < -L
                if cos(K * (-L)) > 0
                    c = cos(K * -L) / exp(k * -L);
                    y(j) = c * exp(k * x(j));
                else
                    c = -cos(K * -L) / exp(k * -L);
                    y(j) = -c * exp(k * x(j));
                end
            end
        end

    end
```

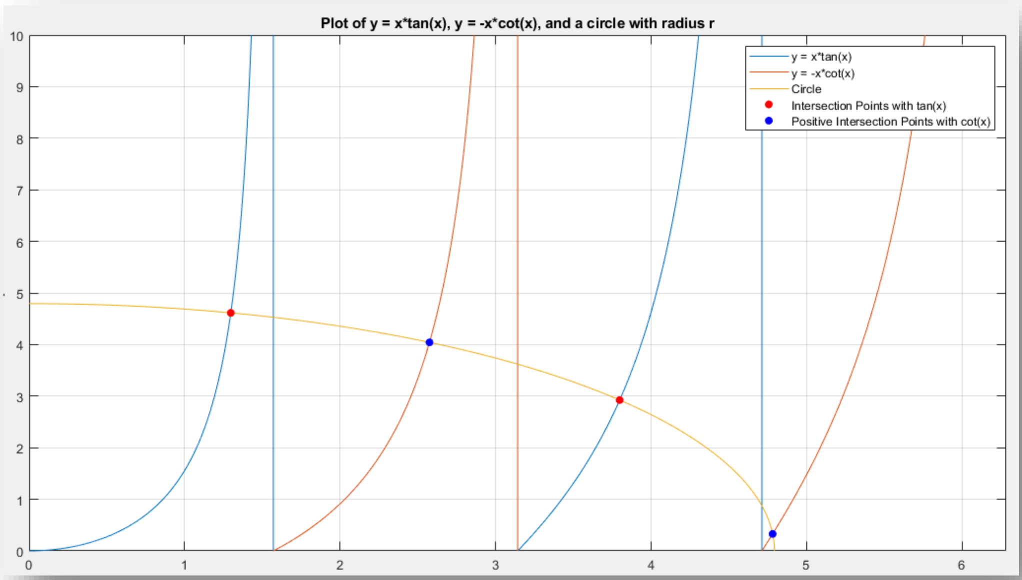


```
    plot(x, y, 'b-', 'LineWidth', 2);

    if mod(i, 2) == 0
        title(['Even Subplot ' num2str(i/2)]);
    else
        title(['Odd Subplot ' num2str((i+1)/2)]);
    end

    xlabel('x');
    ylabel('ψ(x)');
end
```

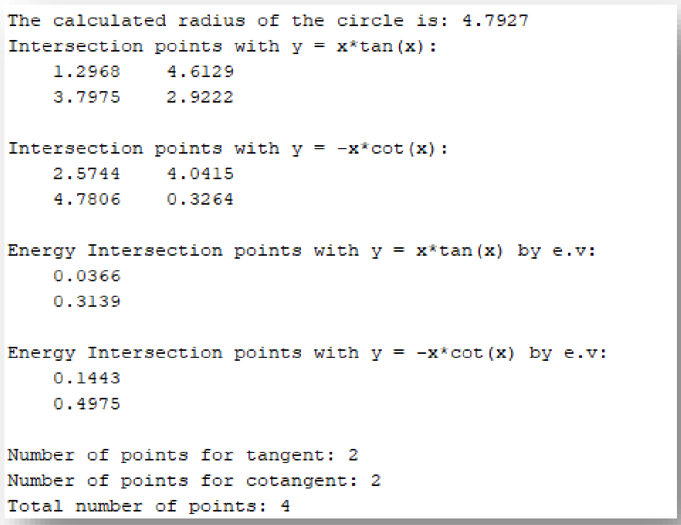
13. **Result** 😊

Now let's check the final results:

As it is known, based on the data, the following figure appears because the circle has been drawn and the tan and cot graphs and its intersection have been checked and drawn.

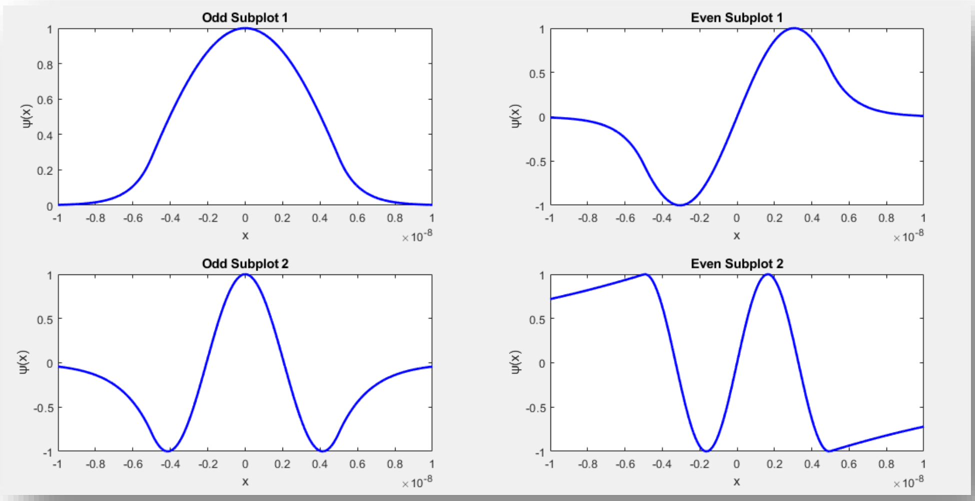


The radius of the circle, the intersection points of the circle with each graph, the number of intersections with each graph, and the total number of points and energy of each are determined:



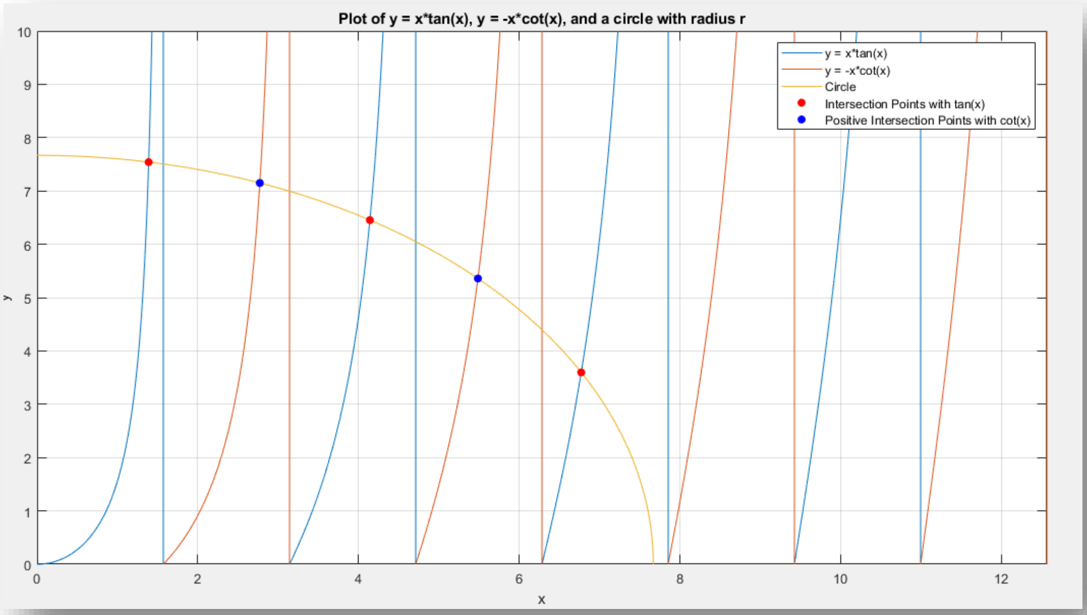
```
The calculated radius of the circle is: 4.7927
Intersection points with y = x*tan(x):
    1.2968    4.6129
    3.7975    2.9222

Intersection points with y = -x*cot(x):
    2.5744    4.0415
    4.7806    0.3264

Energy Intersection points with y = x*tan(x) by e.v:
    0.0366
    0.3139

Energy Intersection points with y = -x*cot(x) by e.v:
    0.1443
    0.4975

Number of points for tangent: 2
Number of points for cotangent: 2
Total number of points: 4
```

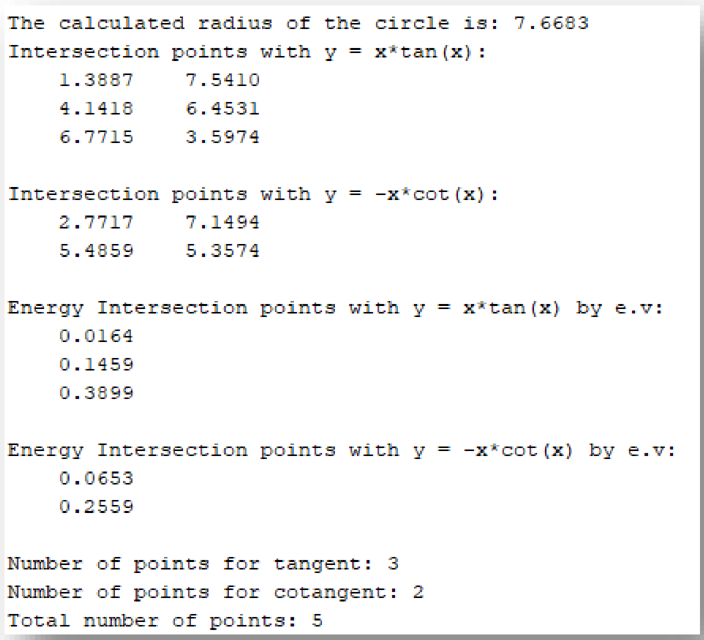
The wave function is as follows:



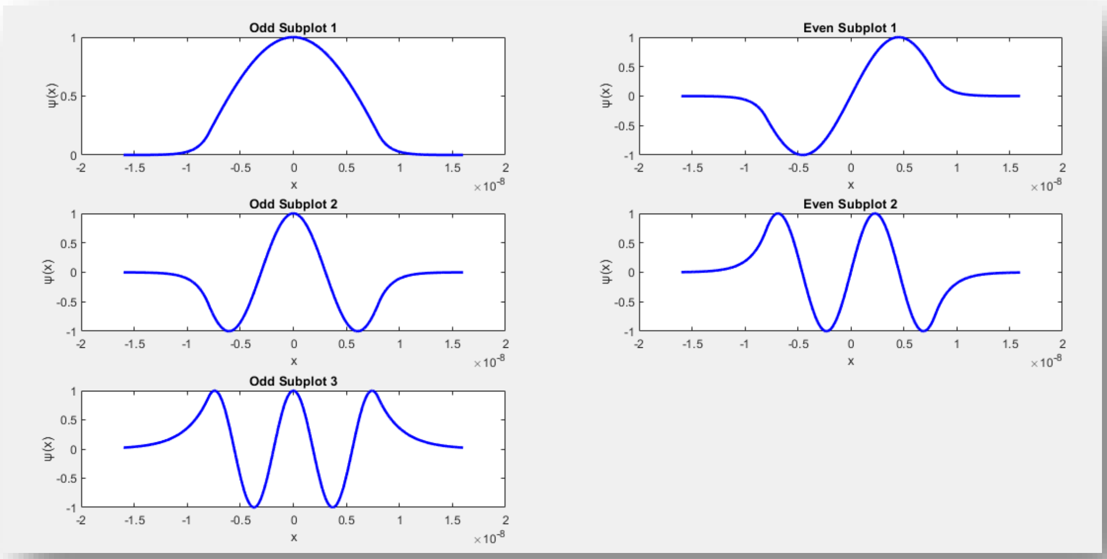
14. Analysis:

Now, if we change the length, the graph and results will be as follows: ($L = 8\ nm$)



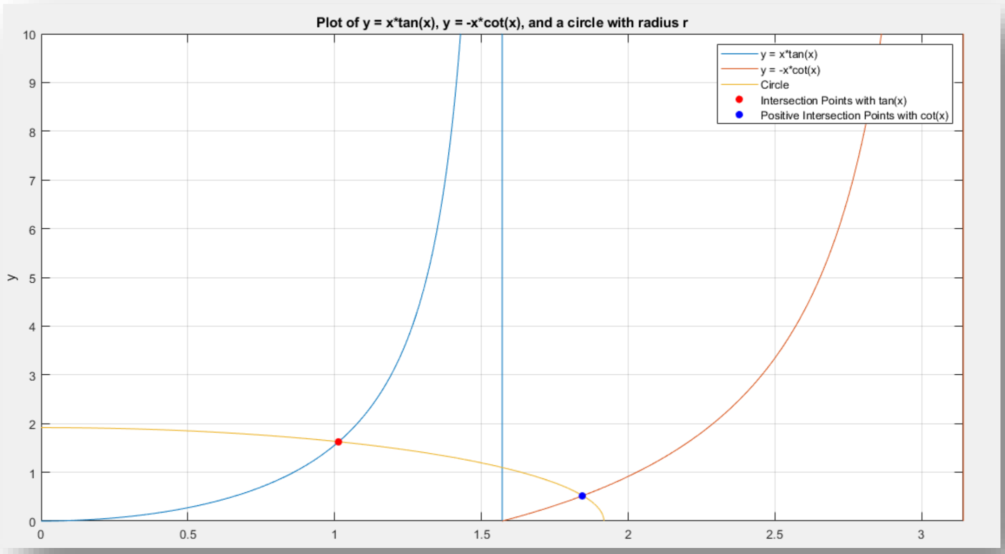


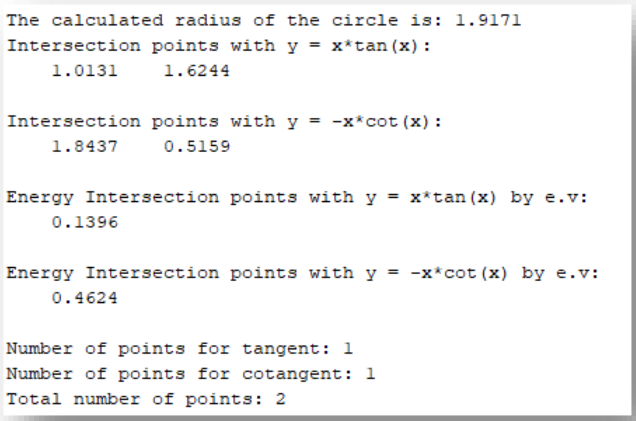
```
The calculated radius of the circle is: 7.6683
Intersection points with y = x*tan(x):
    1.3887    7.5410
    4.1418    6.4531
    6.7715    3.5974

Intersection points with y = -x*cot(x):
    2.7717    7.1494
    5.4859    5.3574

Energy Intersection points with y = x*tan(x) by e.v:
    0.0164
    0.1459
    0.3899

Energy Intersection points with y = -x*cot(x) by e.v:
    0.0653
    0.2559

Number of points for tangent: 3
Number of points for cotangent: 2
Total number of points: 5
```



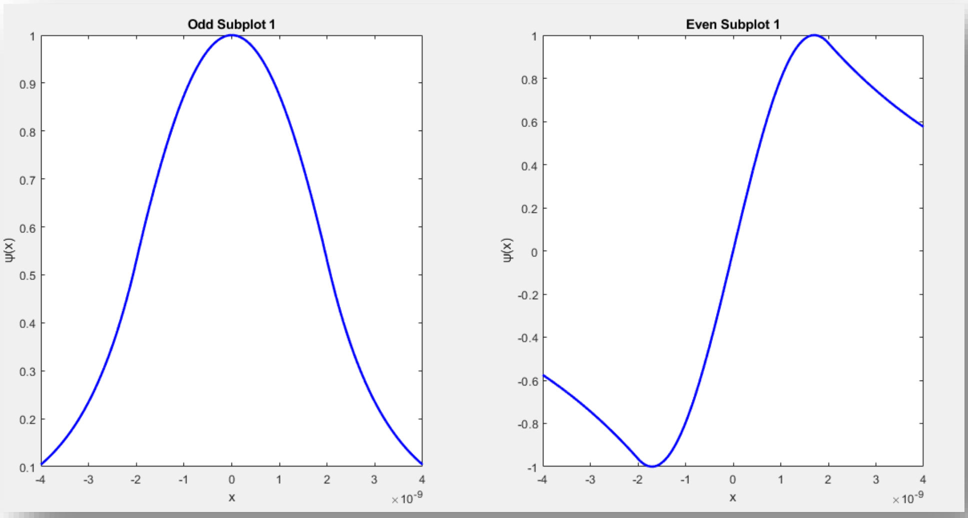
As we know that with the increase in the width of the well, the energy levels and the wave function increase.

Now, if we change the length, the graph and results will be as follows: $(L = 2\ nm)$



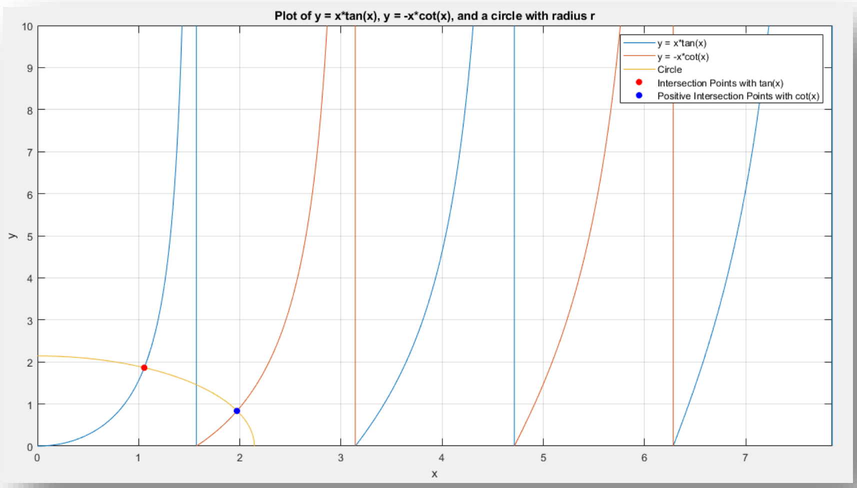


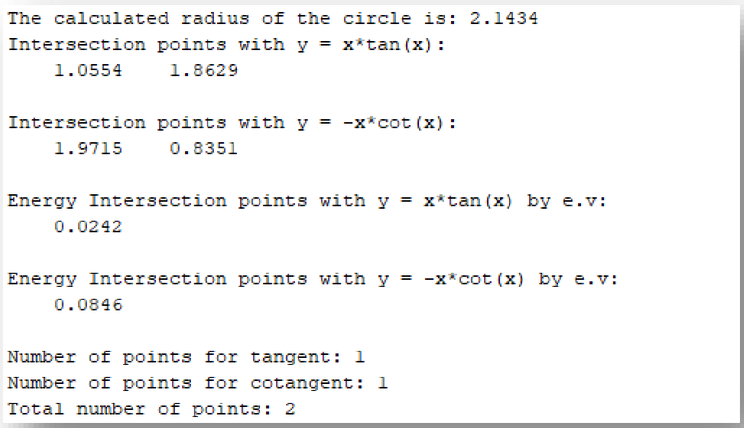
```
The calculated radius of the circle is: 1.9171
Intersection points with y = x*tan(x):
    1.0131    1.6244

Intersection points with y = -x*cot(x):
    1.8437    0.5159

Energy Intersection points with y = x*tan(x) by e.v:
    0.1396

Energy Intersection points with y = -x*cot(x) by e.v:
    0.4624

Number of points for tangent: 1
Number of points for cotangent: 1
Total number of points: 2
```



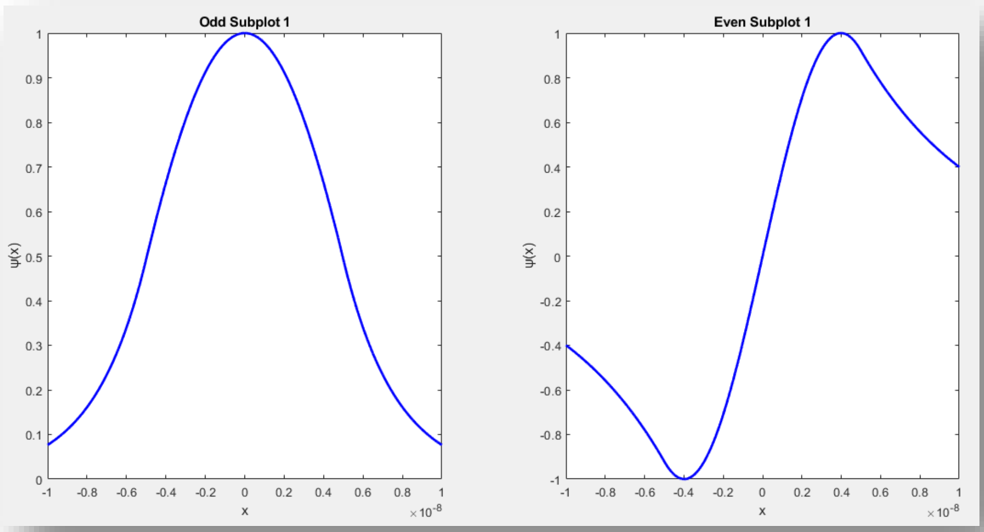
As we know that with the reduction of the width of the well, the energy levels and the wave function decrease.

Now, if we change the U_0, the graph and results will be as follows: $(U_0 = 0.1)$



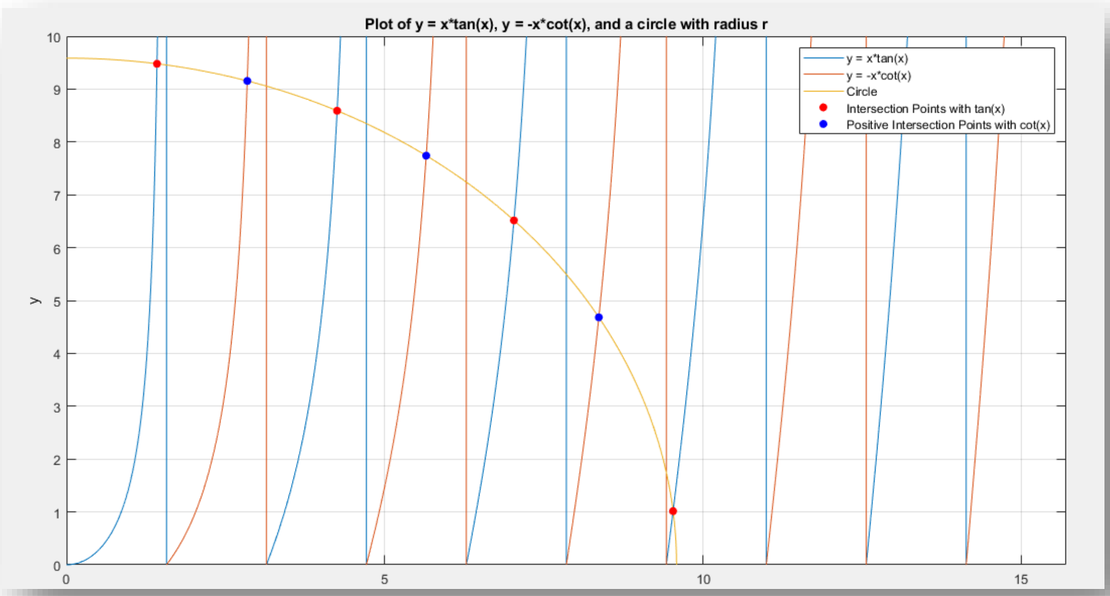


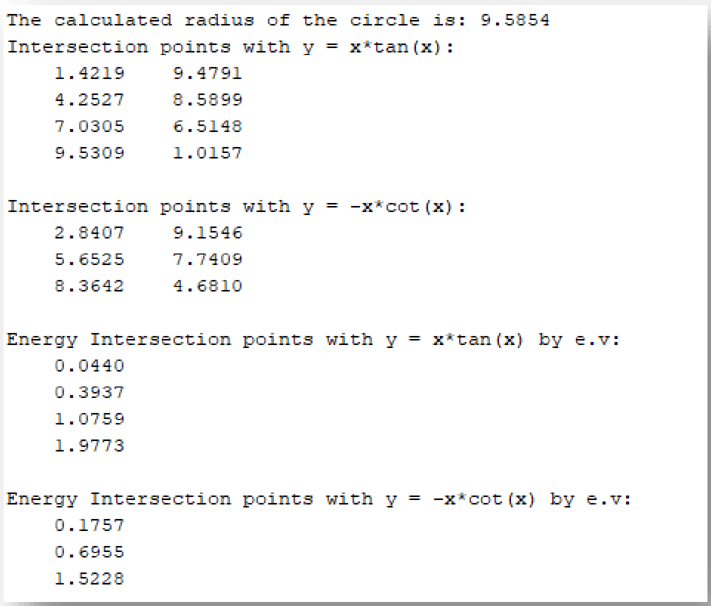
```
The calculated radius of the circle is: 2.1434
Intersection points with y = x*tan(x):
    1.0554    1.8629

Intersection points with y = -x*cot(x):
    1.9715    0.8351

Energy Intersection points with y = x*tan(x) by e.v:
    0.0242

Energy Intersection points with y = -x*cot(x) by e.v:
    0.0846

Number of points for tangent: 1
Number of points for cotangent: 1
Total number of points: 2
```



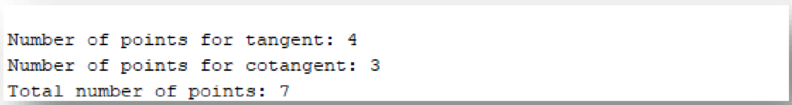
As we know that with the decrease in the height of the well, the energy levels and wave functions decrease.

Now, if we change the U_0, the graph and results will be as follows: $(U_0 = 2)$

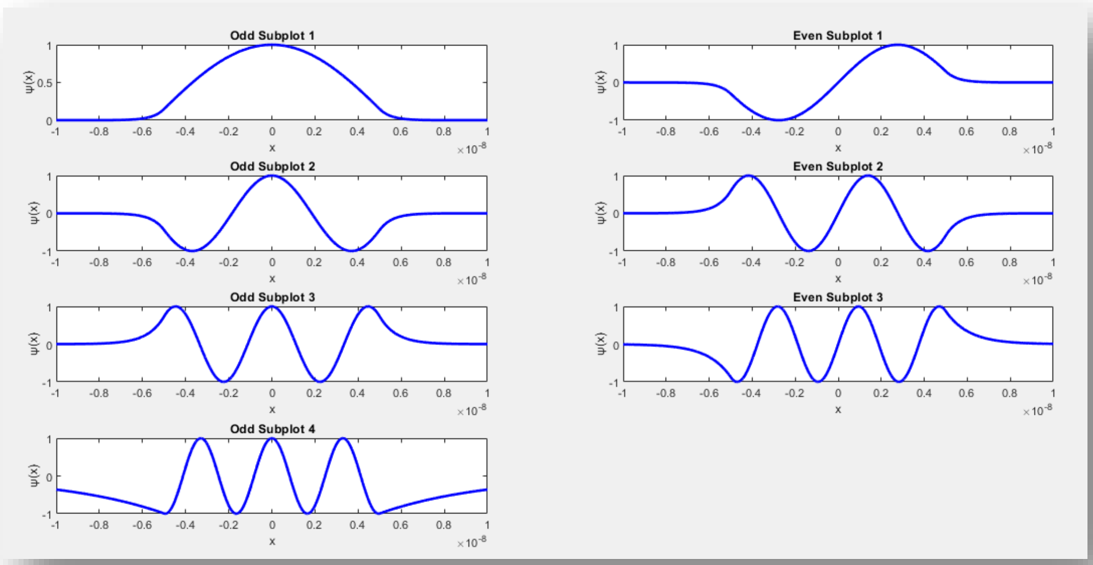




```
The calculated radius of the circle is: 9.5854
Intersection points with y = x*tan(x):
    1.4219    9.4791
    4.2527    8.5899
    7.0305    6.5148
    9.5309    1.0157

Intersection points with y = -x*cot(x):
    2.8407    9.1546
    5.6525    7.7409
    8.3642    4.6810

Energy Intersection points with y = x*tan(x) by e.v:
    0.0440
    0.3937
    1.0759
    1.9773

Energy Intersection points with y = -x*cot(x) by e.v:
    0.1757
    0.6955
    1.5228
```



```
Number of points for tangent: 4
Number of points for cotangent: 3
Total number of points: 7
```



As we know that with the increase in the height of the well, the energy levels and the function of the waves increase.