Problem 1 (Compute square roots)

```
a = 5;
 % Array to store x values
f = zeros(6,1);
% Initial guess
f(1) = 0.5;

% Perform iterations
for n = 1:5
    f(n+1) = (f(n)+a/f(n))/2;
end

result = f(end)
```

```
result = 2.2361
```

```
fprintf('Approximate square root of %d: %f\n', a, result);
```

```
Approximate square root of 5: 2.236070
```

```


% display the results
iteration = (0:5)';
x_values = f;
true_sqrt_5 = sqrt(5)*ones(6,1);

% display table
table(iteration, x_values, true_sqrt_5,...
    'VariableNames', {'Iteration','Computed_Value','True_Value'})
```

ans = 6×3 table

	Iteration	Computed_Value	True_Value
1	0	0.5000	2.2361
2	1	5.2500	2.2361
3	2	3.1012	2.2361
4	3	2.3567	2.2361
5	4	2.2392	2.2361
6	5	2.2361	2.2361

Problem 2 (Using fzero)

```
clc
% variables
ED = 10e-4;% relative roughness
ReD = 3.3e5;% Reynolds number
tolerance = 1e-8;

% colebrook equation as function of f
```

```
colebrook_equation = @(f) 1/sqrt(f) + 2*log10((ED/3.7) + (2.51/
(ReD*sqrt(f))));

guess = 0.0109; % initial guess needs to be smaller than 0.01..
%tolerance using optimset
options = optimset('TolX', tolerance);
% find friction factor with fzero
friction_factor = fzero(colebrook_equation, guess, options);

% display result
fprintf('Friction Factor f: %.6f\n', friction_factor);
```

```
Friction Factor f: 0.020522
```

Problem 2 (part. b)

```
clc
% call function
optimum_z = find_optimum_damping_ratio();
fprintf('Optimum damping ratio z: %.6f\n', optimum_z);
```

```
Optimum damping ratio z: 0.403973
```

Problem 3 (Freezing water mains)

```
clc
% Use fzero to find x
x_solution = fzero(@freezing_condition, 1);
fprintf('The pipe should be buried at about %.2f meters deep to avoid
freezing for at least 30 days.\n', x_solution);
```

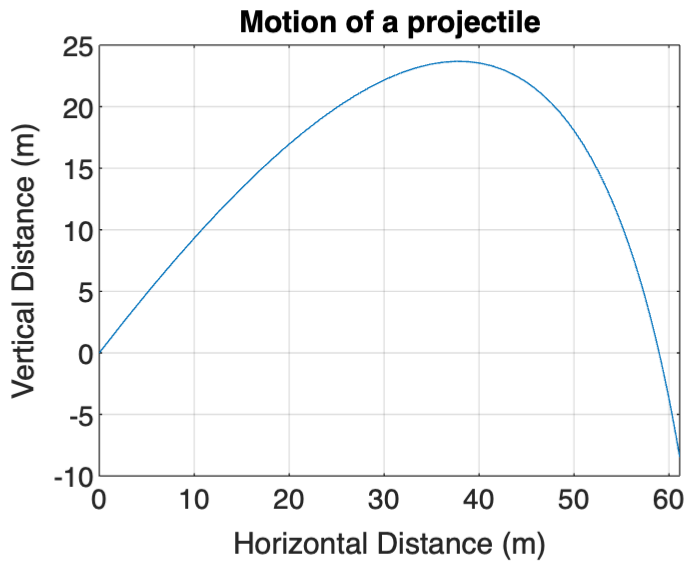
```
The pipe should be buried at about 0.97 meters deep to avoid freezing for at least 30 days.
```

Problem 4 (Motion of a projectile)

```
clc
% variables
v0 = 40; %m/s initial v
theta = pi/4; %radians
k = 2; %kg/s
m = 5; %kg
t = 0:0.1:5; %time 0-5 intervals of 0.01
alpha_1 = k/m;
colebrook_eq = 9.81; %m/s gravitational accelaration

% given equations
x_t = (v0/alpha_1)*cos(theta)*(1-exp(-alpha_1*t));
y_t = ((colebrook_eq/alpha_1^2)+((v0/alpha_1)*sin(theta)))*(1-exp(-
alpha_1*t))-((colebrook_eq*t)/alpha_1);
```

```
% Plot the projectile's motion for 0<=t<=5
figure;
plot(x_t, y_t,'-');
title('Motion of a projectile');
xlabel('Horizontal Distance (m)');
ylabel('Vertical Distance (m)');
% zlabel('Time (s)')
grid on;
```



```


%find the minimum of the vetical component y_t = 0
 ground_impact0 = fminbnd(@(t) ((colebrook_eq/alpha_1^2)+((v0/
alpha_1)*sin(theta)))*(1-exp(-alpha_1*t))-((colebrook_eq*t)/alpha_1), 0, 5);

% Calculate the horizontal component x_t at that time
 x_ground = (v0/alpha_1)*cos(theta)*(1-exp(-alpha_1*ground_impact0));

% Display the results
fprintf('Horizontal position at 5 seconds: %.3f meters\n', x_ground);
```

```
Horizontal position at 5 seconds: 61.141 meters
```

Problem 5 (Optimization)

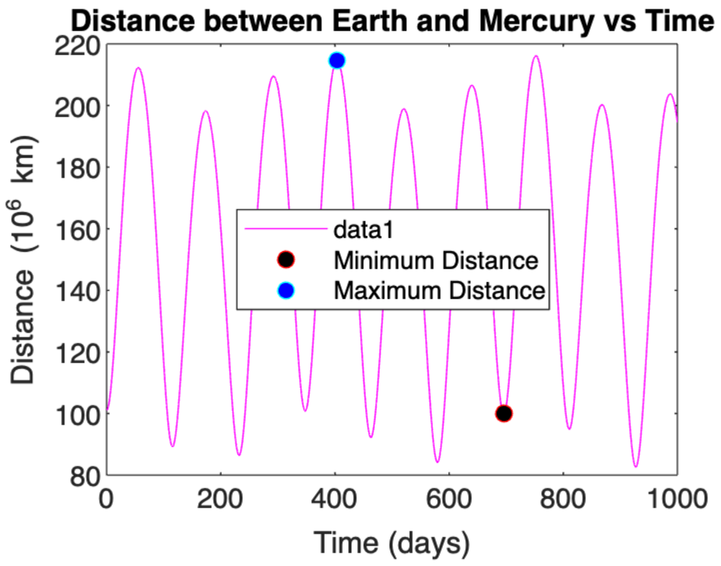
```
clc
%use fminbnd for min and max
t_min = fminbnd(@distance_earth_mercury, 0, 1000);
t_max = fminbnd(@(t) -distance_earth_mercury(t), 0, 1000);

%time between 0 and 1000
time = linspace(0,1000,1000);
%distances for each element of time array
```

```
distances = arrayfun(@distance_earth_mercury, time);

% Plot
figure;
plot(time,distances, '-', Color='magenta');
xlabel('Time (days)');
ylabel('Distance (10^6 km)');
title('Distance between Earth and Mercury vs Time');

%Plot minimum and maximum distances
hold on;
plot(t_min,distance_earth_mercury(t_min), 'ro', 'MarkerFaceColor','k',
'DisplayName', 'Minimum Distance');
plot(t_max,distance_earth_mercury(t_max), 'co', 'MarkerFaceColor','blue',
'DisplayName', 'Maximum Distance');
legend('Location','Best');
hold off;
```



Function (Problem 5)

```
function distance = distance_earth_mercury(t)
    % given parameters for Earth and Mercury
    xm = -11.9084+57.9117*cos(2*pi*t/87.97);
    ym = 56.6741*sin(2*pi*t/87.97);
    xe = -2.4987+149.6041*cos(2*pi*t/365.25);
    ye = 149.5832*sin(2*pi*t/365.25);

    % calculate using distance formula
    distance = sqrt((xe-xm)^2+(ye-ym)^2);
end
```

Function (Problem 3)

```
function equation = freezing_condition(x)
    Ts = -4.5;% celcius
    Ti = 18.3;% celcius
    alpha = 0.138e-6;% m^2/s
    t = 30*24*60*60; % 30 days in seconds
    equation = (T(x,t)-Ts)/(Ti-Ts)-erf(x/(2*sqrt(alpha*t)));
end

function temperature = T(x,t)
    Ts = -4.5; % celcius
    Ti = 18.3; % celcius
    alpha = 0.138e-6;% m^2/s
    temperature = Ts+(Ti-Ts)*erf(x/(2*sqrt(alpha*t)));
end
```

Function (Problem 2 part. b)

```
function optimum_damping_ratio = find_optimum_damping_ratio()
    % define function
    h = @(z) cos(4*z*sqrt(1-z^2))+1-8*z^2+8*z^4;
    tolerance1 = 1e-12;
    % optimset for fzero
    x_1 = optimset('TolX', tolerance1);
    % fzero with a range of [0, 0.5]
    optimum_damping_ratio = fzero(h,[0, 0.5],x_1);
end
```