Numerical methods for engineers - HW 1
Leeel and Dror

1_a

```
function norms = MyDist_a(P,P0)
   norms = sqrt((P(:,1)-P0(1)).^2+(P(:,2)-P0(2)).^2)
end
```

1_b

```
function norms = MyDist_b(P,P0)
norms = [];
for i = 1:size(P,1)
   R = sqrt((P(i,1)-P0(1))^2+(P(i,2)-P0(2))^2);
   norms(end+1)  = R;
end
End
```

1_c

MyDist_a: 0.006440 seconds
MyDist_b: 0.041396 seconds

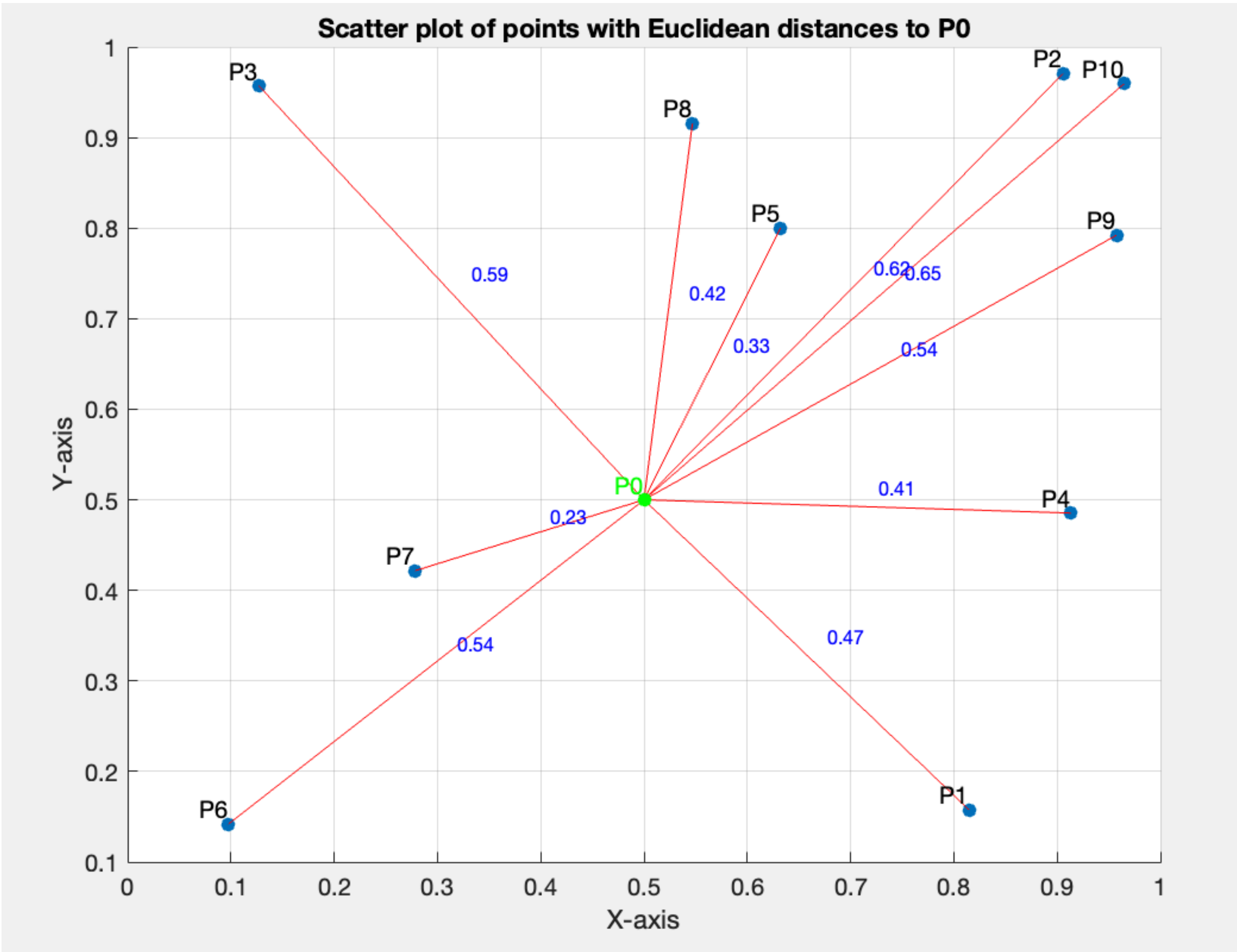
1_d

```
function [] = MyPlot(P, P0)
 norms = MyDist_a(P, P0);
```

```
    scatter(P(:,1), P(:,2), 'filled');
    hold on;

   % Plot lines from each point in P to P0
   for i = 1:size(P, 1)
       plot([P(i, 1), P0(1)], [P(i, 2), P0(2)], 'r');
       text(P(i, 1), P(i, 2), ['P' num2str(i)], 'VerticalAlignment',
     'bottom', 'HorizontalAlignment', 'right');

       % Annotate the distance with a smaller offset
       offset = 0.02;
       rounded_distance = round(norms(i), 2); % Round the distance to
     the hundredths place
       text((P(i, 1) + P0(1)) / 2 + offset, ...
            (P(i, 2) + P0(2)) / 2 + offset, ...
            num2str(rounded_distance, '%.2f'), 'Color', 'blue',
     'FontSize', 8);
   end

   % Plot P0
   scatter(P0(1), P0(2), 'filled', 'MarkerFaceColor', 'g');
   text(P0(1), P0(2), 'P0', 'VerticalAlignment', 'bottom',
     'HorizontalAlignment', 'right', 'Color', 'green');

   % Set axis labels and title
   xlabel('X-axis');
   ylabel('Y-axis');
   title('Scatter plot of points with Euclidean distances to P0');
   grid on;
   hold off;
end
```

Discussion and conclusion Q1:

The performance comparison between functions, MyDist_a and MyDist_b, shows differences in execution efficiency. MyDist_a uses vectorized operations, so MATLAB processes the whole dataset at once which is faster in one order of magnitude (0.006440 seconds) compared to the loop-based in MyDist_b (0.041396 seconds). This example shows the power of vectorization in MATLAB.



Graph 1.1 - Scatter plot of points with Euclidean distances to point P0

Graph 1.1 display points and their Euclidean distances to a reference point.

2_a

```
syms x y;
f = sin(x) * log(x * y);
a_x = 1;
a_y = 1;
f_a = subs(f, {x, y}, {a_x, a_y});
f_x = diff(f, x);
f_y = diff(f, y);
f_x_a = subs(f_x, {x, y}, {a_x, a_y});
f_y_a = subs(f_y, {x, y}, {a_x, a_y});
f_xx_a = subs(diff(f_x, x), {x, y}, {a_x, a_y});
f_yy_a = subs(diff(f_y, y), {x, y}, {a_x, a_y});
f_xy_a = subs(diff(f_x, y), {x, y}, {a_x, a_y});
f_xxx_a = subs(diff(f_xx_a, x), {x, y}, {a_x, a_y});
```

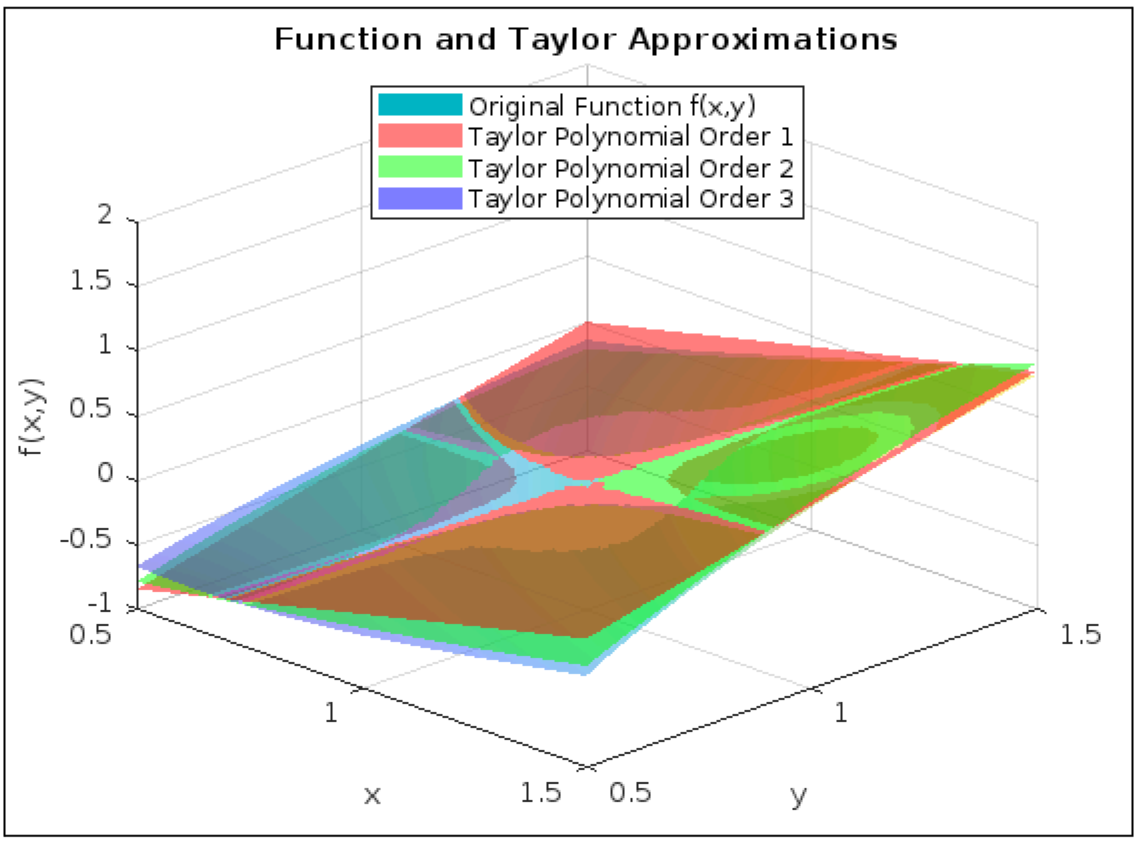
```
f_yyy_a = subs(diff(f_yy_a, y), {x, y}, {a_x, a_y});
f_xxy_a = subs(diff(f_xx_a, y), {x, y}, {a_x, a_y});
f_xyy_a = subs(diff(f_xy_a, y), {x, y}, {a_x, a_y});
f_xyx_a = subs(diff(f_xy_a, x), {x, y}, {a_x, a_y});
% Taylor Polynomial of order 1
T1 = f_a + f_x_a * (x - a_x) + f_y_a * (y - a_y);
% Taylor Polynomial of order 2
T2 = T1 + (f_xx_a / 2) * (x - a_x)^2 + (f_yy_a / 2) * (y - a_y)^2 + f_xy_a * (x
            - a_x) * (y - a_y);
% Taylor Polynomial of order 3
T3 = T2 + (f_xxx_a / 6) * (x - a_x)^3 + (f_yyy_a / 6) * (y - a_y)^3 + ...
   (f_xxy_a / 2) * (x - a_x)^2 * (y - a_y) + (f_xyy_a / 2) * (x - a_x) * (y -
            a_y)^2 + ...
   (f_xyx_a / 6) * (x - a_x)^2 * (y - a_y);
% Results
disp('Taylor Polynomial of Order 1:');
pretty(T1)
disp('Taylor Polynomial of Order 2:');
pretty(T2)
disp('Taylor Polynomial of Order 3:');
pretty(T3)
```

2_b

Continue from 2_a

```
% Change symbolics to functions for plotting
f_handle = matlabFunction(f);
T1_handle = matlabFunction(T1);
T2_handle = matlabFunction(T2);
T3_handle = matlabFunction(T3);

% Create a grid
[x_vals, y_vals] = meshgrid(0.5:0.01:1.5, 0.5:0.01:1.5);

% Functions on the grid
f_vals = f_handle(x_vals, y_vals);
T1_vals = T1_handle(x_vals, y_vals);
T2_vals = T2_handle(x_vals, y_vals);
T3_vals = T3_handle(x_vals, y_vals);

%Plotting
figure;
hold on;
% Original function
surf(x_vals, y_vals, f_vals, 'FaceColor', 'interp', 'FaceAlpha', 0.5,
'EdgeColor', 'none', 'DisplayName', 'Original Function f(x,y)');
% Taylor approximations
```

```
surf(x_vals, y_vals, T1_vals, 'FaceColor', 'red', 'FaceAlpha', 0.5,
'EdgeColor', 'none', 'DisplayName', 'Taylor Polynomial Order 1');
surf(x_vals, y_vals, T2_vals, 'FaceColor', 'green', 'FaceAlpha', 0.5,
'EdgeColor', 'none', 'DisplayName', 'Taylor Polynomial Order 2');
surf(x_vals, y_vals, T3_vals, 'FaceColor', 'blue', 'FaceAlpha', 0.5,
'EdgeColor', 'none', 'DisplayName', 'Taylor Polynomial Order 3');
% graph setting
view(45, 30);
xlabel('x');
ylabel('y');
zlabel('f(x,y)');
title('Function and Taylor Approximations');
legend('Location', 'Best');
xlim([0.5, 1.5]);
ylim([0.5, 1.5]);
zlim([-1, 2]);
grid on;
hold off;
```



Graph 2.1 original function and 1, 2 and 3 Taylor polynomial orders.

Discussion and conclusion Q2:

Graph 2.1 shows the original function and its Taylor orders and illustrates how the accuracy improves with higher order equations.
The first-order polynomial is linear, the second-order (T2) and third-order (T3) are polynomials. The graph shows that higher order equations match more with the original function, particularly near the point (1, 1).

3

```
clc;
x = [1; 1];
x_prev = x;
e = 1e-5;
IterN = 0;
while norm(x - x_prev) > e
   x_prev = x;

   % System of equations
   F = [x(1)^2 + x(2)^2 - 4;
        exp(x(1)) + x(2) - 1];

   % Jacobian matrix
   J = [2*x(1), 2*x(2);
        exp(x(1)), 1];

   % Newton's method
   dx = -J \ F;
   x = x + dx; % Update x
   IterN = IterN + 1; % Iteration count
end
% Results
disp('Solution:');
disp(x);
disp('Number of iterations:');
disp(IterN);
```

Discussion and conclusion Q3:
Two main changes were made

1) In the original code the tolerance 1e5 was very large, so the stopping condition couldn't be satisfied. Changing the tolerance to 1e-5 the algorithm iterates until the solution is accurate. It requires more iterations but makes a more precise solution.
2) The constant values changed from -4 and -1 to 1 and 4. Because the equations are trying to solve for roots, the sum of squares for -4 and -1 are not likely to have real numbers solutions. By doing so, we change the equations targets, so the algorithm can converge on solutions that are realistic for the revised system.

4

```
Start
Input x
If x < 10 Then
              If x < 5 Then
                    Print x
                Else
                    x = 5
                    Print x
                End If
             If x < 50 Then
                      Print x
  Else
        x = x - 5
        Print x
    End If
Else
   End If
End
```

References:

Taylor series in several variables,
https://en.wikipedia.org/wiki/Taylor_series#Taylor_series_in_several_variables

Newton Raphson method for a system of non-linear equations
https://www.mathworks.com/matlabcentral/answers/1911085-newton-raphson-method-for-a-system-of-non-linear-equations