Home Work 3

Muthu Ram Kumar Avichi (1010188967)

Q1 - Matlab Code:

```
syms x1 x2 x3
f1 = x1^2 + x2^2 + x3^2;
newtonsMethodBacktracking(f1, [1; 1; 1], 0.5, 0.5, 1000);
f2 = x1^2 + 2*x2^2 - 2*x1*x2 - 2*x2;
newtonsMethodBacktracking(f2, [0; 0], 0.5, 0.5, 1000);
f3 = 100*(x2 - x1^2)^2 + (1 - x1)^2;
newtonsMethodBacktracking(f3, [-1.2; 1], 0.5, 0.5, 1000);
f4 = (x1+x2)^4 + x2^2;
newtonsMethodBacktracking(f4, [2; -2], 0.5, 0.5, 1000);
f5_1 = (x1 - 1)^2 + (x2 - 1)^2 + (x1^2 + x2^2 - 0.25)^2;
newtonsMethodBacktracking(f5_1, [1; -1], 0.5, 0.5, 1000);
f5_2 = (x1 - 1)^2 + (x2 - 1)^2 + 10*(x1^2 + x2^2 - 0.25)^2;
newtonsMethodBacktracking(f5_2, [1; -1], 0.5, 0.5, 1000);
f5_3 = (x1 - 1)^2 + (x2 - 1)^2 + 100*(x1^2 + x2^2 - 0.25)^2;
newtonsMethodBacktracking(f5_3, [1; -1], 0.5, 0.5, 1000);
function newtonsMethodBacktracking(f, x0, beta, gamma, maxIterations)
   syms x1 x2 x3
   if length(x0) == 3
       x = [x1 x2 x3];
   elseif length(x0) == 2
       x = [x1 x2];
   end
   % Calculate gradient of f
   grad_f = gradient(f, x);

   % Constants for backtracking
   alpha_k = 1;

   % Initialize
   x_k = x0;
   iteration_counter = 1;

   % Perform Newton's backtracking algorithm
   while iteration_counter <= maxIterations
       % Calculate gradient at current x_k
       grad_val = vpa(subs(grad_f, x, x_k.'), 200);
       fx_val = vpa(subs(f, x, x_k.'), 200);

       if norm(grad_val)/(1+norm(fx_val)) <= 10e-5
```

```
            break;
        end
        d_k = vpa(-grad_val, 200);  % Assuming Hessian is identity for gradient
descent
        f_xk = vpa(subs(f, x, x_k.'), 200);
        while true
            x_k_plus_1 = x_k + alpha_k * d_k;
            f_xk_plus_1 = vpa(subs(f, x, x_k_plus_1.'), 200);
            rhs_backtracking = vpa(gamma * alpha_k * (grad_val.') * d_k, 200);
            if f_xk - f_xk_plus_1 >= -rhs_backtracking
                break;
            else
                alpha_k = beta * alpha_k;
            end
        end
        x_k = vpa(x_k_plus_1, 200);
        iteration_counter = iteration_counter + 1;
        % Store current iteration result
        iterations(:, iteration_counter) = [x_k; d_k; alpha_k];
    end
    disp(['Number of iterations for convergence: ',
num2str(iteration_counter)]);
    disp(['Final solution: x = [', char(x_k.'), ']']);

    % Choose display format based on iteration count
    if iteration_counter > 15
        % Prepare table for the first 10 iterations
        Iteration_fcount = (1:10)';
        X1_First = round(iterations(1, 1:10)', 6);
        X2_First = round(iterations(2, 1:10)', 6);
        d_k_First = round(iterations(3:4, 1:10)', 6);
        alpha_k_First = round(iterations(5, 1:10)', 6);
        T_First = table(Iteration_fcount, X1_First, X2_First, d_k_First,
alpha_k_First);

        % Prepare table for the last 5 iterations
        Iteration_lcount = (iteration_counter-4:iteration_counter)';
        X1_Last = round(iterations(1, end-4:end)', 6);
        X2_Last = round(iterations(2, end-4:end)', 6);
        d_k_Last = round(iterations(3:4, end-4:end)', 6);
        alpha_k_Last = round(iterations(5, end-4:end)', 6);
        T_Last = table(Iteration_lcount, X1_Last, X2_Last, d_k_Last,
alpha_k_Last);

        disp('First 10 iterations:');
        disp(T_First);
        disp('Last 5 iterations:');
        disp(T_Last);
    else
```

```
        % Prepare table for all iterations
        IterationsAll = (1:iteration_counter)';
        X1All = round(iterations(1, :)', 6);
        X2All = round(iterations(2, :)', 6);
        d_k_All = round(iterations(3:4, :)', 6);
        alpha_k_All = round(iterations(5, :)', 6);
        TAll = table(IterationsAll, X1All, X2All, d_k_All, alpha_k_All);

        disp('All iterations:');
        disp(TAll);
    end
end
```

Code Explanation:

- This code contains the results for all five parts of the question. I have incorporated Newton's gradient based method with backtracking to compute step-lengths.
- The initial assumption of alpha = 1, beta = 0.5 and gamma = 0.5 (heuristically).
- I have implemented the stopping condition based on the part (b) question for the entire code and implemented all five conditions using a while loop.
- Theoretically, the gradient based method with backtracking shows convergence similar to the algorithm, except for a few nuances in output. For instance, the variable precision arithmetic in Matlab enables this algorithm to implement results to a precision of 200 decimal places. This gives much more accurate results that converge faster compared to theoretical approaches that can restrict the speed of convergence, or at times show diverging results due to rounding errors.

Results:

Q1-(d)-(1)

```
Number of iterations for convergence: 2
Final solution: x = [[0, 0, 0]]
All iterations:
```

IterationsAll	X1All	X2All	d_k_All		alpha_k_All
1	0	0	0	0	0
2	0	0	0	-2.0	-2.0

Q1-(d)-(2)

Command Window

```
Number of iterations for convergence: 42
Final solution: x = [[0.99980286955315023078583180904388, 0.99987816668362938798964023590088]]
First 10 iterations:
```

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	0	0.5	0	2.0	0.25
3	0.25	0.5	1.0	0	0.25
4	0.375	0.625	0.5	0.5	0.25
5	0.5	0.6875	0.5	0.25	0.25
6	0.59375	0.75	0.375	0.25	0.25
7	0.671875	0.796875	0.3125	0.1875	0.25
8	0.734375	0.835938	0.25	0.15625	0.25
9	0.785156	0.867188	0.203125	0.125	0.25
10	0.826172	0.892578	0.164062	0.101562	0.25

Last 5 iterations:

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
38	0.99954	0.999716	0.000435	0.000269	0.25
39	0.999628	0.99977	0.000352	0.000217	0.25
40	0.999699	0.999814	0.000284	0.000176	0.25
41	0.999756	0.999849	0.00023	0.000142	0.25
42	0.999803	0.999878	0.000186	0.000115	0.25

Q1-(d)-(3)

```
Number of iterations for convergence: 1001
Final solution: x = [[-0.53208813729322818617734654797784, 0.29077448855749259682708231355158]]
First 10 iterations:
```

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	-1.094727	1.042969	215.6	88.0	0.000488
3	-1.059442	1.05815	72.262833	31.091499	0.000488
4	-1.044132	1.064426	31.353812	12.853431	0.000488
5	-1.036878	1.066944	14.858047	5.157287	0.000488
6	-1.033234	1.067742	7.46256	1.634139	0.000488
7	-1.031283	1.067726	3.996062	-0.034071	0.000488
8	-1.030141	1.067317	2.337433	-0.836401	0.000488
9	-1.029391	1.066719	1.535876	-1.225272	0.000488
10	-1.028831	1.066028	1.14659	-1.414522	0.000488

Last 5 iterations:

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
997	-0.534884	0.293768	1.427904	-1.534089	0.000488
998	-0.534186	0.293019	1.429231	-1.533545	0.000488
999	-0.533487	0.292271	1.43056	-1.532999	0.000488
1000	-0.532788	0.291522	1.431891	-1.532449	0.000488
1001	-0.532088	0.290774	1.433224	-1.531896	0.000488

Q1-(d)-(4)

```
Number of iterations for convergence: 1001
Final solution: x = [[0.069317575759854400976833975256087, -0.00066574218045044521698656210724373]]
First 10 iterations:
```

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	2.0	-1.0	0	4.0	0.25
3	1.875	-1.0625	-4.0	-2.0	0.03125
4	1.807953	-1.063141	-2.145508	-0.020508	0.03125
5	1.756305	-1.048342	-1.652723	0.473559	0.03125
6	1.71195	-1.027176	-1.419358	0.677326	0.03125
7	1.671813	-1.003115	-1.284408	0.769943	0.03125
8	1.634436	-0.977797	-1.19605	0.81018	0.03125
9	1.599045	-0.952075	-1.132506	0.823088	0.03125
10	1.565195	-0.926421	-1.083209	0.820942	0.03125

```
Last 5 iterations:
```

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
997	0.06948	-0.00067	-0.001305	0.000038	0.03125
998	0.069439	-0.000669	-0.001303	0.000038	0.03125
999	0.069399	-0.000668	-0.001301	0.000038	0.03125
1000	0.069358	-0.000667	-0.001299	0.000037	0.03125
1001	0.069318	-0.000666	-0.001296	0.000037	0.03125

Q1-(d)-(5)-(a)

Command Window

```
Number of iterations for convergence: 90
Final solution: x = [[0.56411523230150467957436554701942, 0.56405866233570277641266656152838]]
First 10 iterations:
```

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	0.78125	-0.65625	-7.0	11.0	0.03125
3	0.717674	-0.487846	-2.034424	5.388916	0.03125
4	0.690191	-0.36418	-0.879454	3.957338	0.03125
5	0.678583	-0.262576	-0.371472	3.251308	0.03125
6	0.67497	-0.174494	-0.115607	2.81863	0.03125
7	0.67537	-0.09594	0.012799	2.513733	0.03125
8	0.677481	-0.024861	0.067552	2.274514	0.03125
9	0.679889	0.039844	0.077041	2.070566	0.03125
10	0.681723	0.098789	0.058684	1.886232	0.03125

```
Last 5 iterations:
```

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
86	0.564132	0.564042	-0.00018	0.00018	0.03125
87	0.564127	0.564047	-0.00016	0.00016	0.03125
88	0.564123	0.564051	-0.000143	0.000143	0.03125
89	0.564119	0.564055	-0.000127	0.000127	0.03125
90	0.564115	0.564059	-0.000113	0.000113	0.03125

Q1-(d)-(5)-(b)

Command Window

Number of iterations for convergence: 529

Final solution: x = [[0.40263476913368621815180537640316, 0.40258526886212490210156277151816]]

First 10 iterations:

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	0.726562	-0.710938	-70.0	74.0	0.003906
3	0.639771	-0.610556	-22.218513	25.697685	0.003906
4	0.589396	-0.547213	-12.89608	16.215842	0.003906
5	0.556059	-0.501195	-8.534386	11.780435	0.003906
6	0.532558	-0.46516	-6.0161	9.225196	0.003906
7	0.515408	-0.435543	-4.390523	7.58176	0.003906
8	0.502657	-0.410354	-3.264234	6.44852	0.003906
9	0.493108	-0.388368	-2.444576	5.628416	0.003906
10	0.485974	-0.368784	-1.826216	5.013497	0.003906

Last 5 iterations:

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
525	0.402637	0.402583	-0.000136	0.000136	0.003906
526	0.402636	0.402584	-0.000133	0.000133	0.003906
527	0.402636	0.402584	-0.00013	0.00013	0.003906
528	0.402635	0.402585	-0.000128	0.000128	0.003906
529	0.402635	0.402585	-0.000125	0.000125	0.003906

Q1-(d)-(5)-(c)

Command Window

Number of iterations for convergence: 1001

Final solution: x = [[0.40356591041640511680961249232907, 0.30969309140572952674615247858938]]

First 10 iterations:

Iteration_fcount	X1_First	X2_First	d_k_First		alpha_k_First
1	0	0	0	0	0
2	0.658203	-0.65625	-700.0	704.0	0.000488
3	0.579617	-0.575947	-160.943559	164.460047	0.000488
4	0.532745	-0.527424	-95.995117	99.374608	0.000488
5	0.500738	-0.493794	-65.550674	68.876074	0.000488
6	0.477306	-0.468747	-47.987691	51.294481	0.000488
7	0.459401	-0.449227	-36.67045	39.977042	0.000488
8	0.445316	-0.433523	-28.844934	32.161887	0.000488
9	0.434008	-0.420587	-23.160192	26.493902	0.000488
10	0.42479	-0.409732	-18.876778	22.231215	0.000488

Last 5 iterations:

Iteration_lcount	X1_Last	X2_Last	d_k_Last		alpha_k_Last
997	0.404005	0.309115	-0.226179	0.297957	0.000488
998	0.403895	0.309261	-0.225679	0.297072	0.000488
999	0.403785	0.309405	-0.225181	0.29619	0.000488
1000	0.403675	0.309549	-0.224683	0.29531	0.000488
1001	0.403566	0.309693	-0.224186	0.294433	0.000488

Q2 (a) - Matlab Code:

```
% Problem set 3 - Question 2 - Part a
% Define symbolic variables
syms x1 x2
```

```matlab
% Define the function f
f = 100*x1^4 + 0.01*x2^4;
% Calculate gradient of f
grad_f = gradient(f, [x1, x2]);
% Calculate Hessian of f
hessian_f = hessian(f, [x1, x2]);
% Initial guess (you may change this based on your problem)
x_k = [1; 1]; % Column vector [x1; x2]
% Display settings
format long
% Maximum iterations
max_iterations = 15;
iteration_counter = 1;
% Store iteration results
iterations = [];
pd_counter = 0; % Counter for positive definite Hessian occurrences
% Calculate first iteration values, to start Newton's method
grad_val = double(subs(grad_f, [x1, x2], x_k.'));
hessian_val = double(subs(hessian_f, [x1, x2], x_k.'));
d_k = -inv(hessian_val)*grad_val; % Equivalent to inv(H)*grad
x_k = x_k + d_k;
iterations(:, iteration_counter) = x_k;
% Check if the initial Hessian is positive definite
if all(eig(hessian_val) > 0)
   pd_counter = pd_counter + 1; % Increment if positive definite
end
% Perform check and continue Newton's iterative method.
while true
   % Break condition (You might need a proper convergence check)
   if norm(grad_val) <= 10e-6
       break;
   end
   iteration_counter = iteration_counter + 1;

   % Calculate gradient and Hessian at current x_k
   grad_val = double(subs(grad_f, [x1, x2], x_k.'));
   hessian_val = double(subs(hessian_f, [x1, x2], x_k.'));

   % Check if the initial Hessian is positive definite
   if all(eig(hessian_val) > 0)
       pd_counter = pd_counter + 1; % Increment if positive definite
   end
   % Calculate d_k
   d_k = -inv(hessian_val)*grad_val; % Equivalent to inv(H)*grad

   % Update x_k
   x_k = x_k + d_k;

   % Store current iteration result
```

```
    iterations(:, iteration_counter) = x_k;
end
% Choose display format based on iteration count
if iteration_counter > 15
    % Prepare table for the first 10 iterations
    Iteration_fcount = (1:10)';
    X1_First = iterations(1, 1:10)';
    X2_First = iterations(2, 1:10)';
    T_First = table(Iteration_fcount, X1_First, X2_First);

    % Prepare table for the last 5 iterations
    Iteration_lcount = (iteration_counter-4:iteration_counter)';
    X1_Last = iterations(1, end-4:end)';
    X2_Last = iterations(2, end-4:end)';
    T_Last = table(Iteration_lcount, X1_Last, X2_Last);

    disp('First 10 iterations:');
    disp(T_First);
    disp('Last 5 iterations:');
    disp(T_Last);
else
    % Prepare table for all iterations
    IterationsAll = (1:iteration_counter)';
    X1All = iterations(1, :)';
    X2All = iterations(2, :)';
    TAll = table(IterationsAll, X1All, X2All);

    disp('All iterations:');
    disp(TAll);
end
disp('Number of iterations for convergence:');
disp(iteration_counter);
if pd_counter == iteration_counter
    disp('Hessian matrices are positive definite in each iteration until
convergence');
else
    disp('Hessian matrices were not always positive definite');
end
```

Q2 (a) Results:

```
First 10 iterations:
    Iteration_fcount         X1_First              X2_First
    ________________    __________________    __________________

           1             0.666666666666667     0.666666666666667
           2             0.444444444444444     0.444444444444444
           3             0.296296296296296     0.296296296296296
           4             0.197530864197531     0.197530864197531
           5             0.131687242798354     0.131687242798354
           6            0.0877914951989026    0.0877914951989026
           7             0.058527663465935    0.0585276634659351
           8            0.0390184423106234    0.0390184423106234
           9            0.0260122948737489    0.0260122948737489
          10            0.0173415299158326    0.0173415299158326

Last 5 iterations:
    Iteration_lcount          X1_Last                X2_Last
    ________________    ___________________    ___________________

          12            0.00770734662925892    0.00770734662925895
          13            0.00513823108617262    0.00513823108617263
          14            0.00342548739078174    0.00342548739078175
          15            0.00228365826052116    0.00228365826052117
          16            0.00152243884034744    0.00152243884034745

Number of iterations for convergence:
    16

Hessian matrices are positive definite in each iteration until convergence
```

Q2 (b) - Matlab Code:

```
% Define symbolic variables
syms x1 x2
% Define the function f
f = 100*x1^4 + 0.01*x2^4;
% Calculate gradient of f
grad_f = gradient(f, [x1, x2]);
% Initial guess
x_k = [1; 1]; % Column vector [x1; x2]
% Constants for backtracking
beta = 0.5;
gamma = 0.5;
alpha_k = 1;
% Maximum iterations
max_iterations = 10000;
iteration_counter = 1;
% Perform Newton's backtracking algorithm
while iteration_counter <= max_iterations
   % Calculate gradient at current x_k
   grad_val = vpa(subs(grad_f, [x1, x2], x_k.'), 200);
   % Calculate d_k
   d_k = vpa(-grad_val, 200);

   if norm(d_k) <= 10e-3
       break;
```

```
    end

    % Calculate f(x_k)
    f_xk = vpa(subs(f, [x1, x2], x_k.'), 200);

    % Compute alpha_k using backtracking line search
    while true
        % Compute x_k_plus_1 and f(x_k_plus_1)
        x_k_plus_1 = x_k + alpha_k*d_k;
        f_xk_plus_1 = vpa(subs(f, [x1, x2], x_k_plus_1.'), 200);

        % Compute RHS of backtracking condition
        rhs_backtracking = vpa(gamma*alpha_k*grad_val.'*d_k, 200);

        % Check backtracking condition
        if f_xk - f_xk_plus_1 >= -rhs_backtracking
            break; % Exit backtracking loop
        else
            alpha_k = beta*alpha_k; % Update alpha_k
        end
    end

    % Update x_k for the next iteration
    x_k = vpa(x_k_plus_1, 200);

    % Increment iteration counter
    iteration_counter = iteration_counter + 1;

end
disp(['Number of iterations for convergence: ', num2str(iteration_counter)]);
disp(['Final solution: x = [', char(x_k(1)), ', ', char(x_k(2)), ']']);
```

Q2 (b) - Results:

```
Number of iterations for convergence: 10001
Final solution: x = [0.011308882711365907373775840461003, 0.74926153441222453254868870131955]
```

In this solution, I have used a max_iteration counter as 10000 and test condition as norm(d_k)<=10^-3. The first condition is met before the second and hence the iteration terminates to yield results.

Q2 (c) - Code:

```
% Problem set 3 - Question 2 - Part c
% Define symbolic variables
syms x1 x2
```

```matlab
% Define the function f
f = sqrt(x1^2+1) + sqrt(x2^2+1);
% Calculate gradient of f
grad_f = gradient(f, [x1, x2]);
% Calculate Hessian of f
hessian_f = hessian(f, [x1, x2]);
% Initial guess (you may change this based on your problem)
x_k = [1; 1]; % Column vector [x1; x2]
% Display settings
format long
% Maximum iterations
max_iterations = 15;
iteration_counter = 1;
% Store iteration results
iterations = [];
pd_counter = 0; % Counter for positive definite Hessian occurrences
% Calculate first iteration values, to start Newton's method
grad_val = vpa(subs(grad_f, [x1, x2], x_k.'), 500);
hessian_val = vpa(subs(hessian_f, [x1, x2], x_k.'), 500);
d_k = vpa(-inv(hessian_val)*grad_val, 500); % Equivalent to inv(H)*grad
x_k = x_k + d_k;
iterations(:, iteration_counter) = x_k;
% Check if the initial Hessian is positive definite
if all(eig(hessian_val) > 0)
   pd_counter = pd_counter + 1; % Increment if positive definite
end
% Perform check and continue Newton's iterative method.
while true
   % Break condition (You might need a proper convergence check)
   if norm(grad_val) <= 10e-6
       break;
   end

   % Check for singularity of Hessian matrix
   if cond(hessian_val) > 1e10
       disp('Hessian matrix is nearly singular. Terminating iteration.');
       break;
   end
   iteration_counter = iteration_counter + 1;

   % Calculate gradient and Hessian at current x_k
   grad_val = vpa(subs(grad_f, [x1, x2], x_k.'), 500);
   hessian_val = vpa(subs(hessian_f, [x1, x2], x_k.'), 500);

   % Check if the initial Hessian is positive definite
   if all(eig(hessian_val) > 0)
       pd_counter = pd_counter + 1; % Increment if positive definite
   end
   % Calculate d_k
```

```
    d_k = vpa(-inv(hessian_val)*grad_val, 500); % Equivalent to inv(H)*grad

    % Update x_k
    x_k = x_k + d_k;

    % Store current iteration result
    iterations(:, iteration_counter) = x_k;

    disp(iteration_counter);
    disp(vpa(subs(f, [x1, x2], x_k.'), 30))
end
disp('Number of iterations for convergence:');
disp(iteration_counter);
```

Q2 (c) - Results:

Command Window

```
2.82842712474619009760337744842

      14998

2.82842712474619009760337744842

      14999

2.82842712474619009760337744842

      15000

2.82842712474619009760337744842

      15001

2.82842712474619009760337744842

      15002

2.82842712474619009760337744842

      15003

fx 2.82842712474619009760337744842
```

The code does not achieve convergence even after 15000 iterations because of the strict convergence criteria. However, as the function output shows, the code produces exactly the same results upto 20 decimal places, which signifies convergence.

Q2 (d) - Code:

```
% Define symbolic variables
syms x1 x2
% Define the function f
f = sqrt(x1^2+1) + sqrt(x2^2+1);
```

```
% Calculate gradient of f
grad_f = gradient(f, [x1, x2]);
% Calculate Hessian of f
hessian_f = hessian(f, [x1, x2]);
% Initial guess
x_k = [1; 1]; % Column vector [x1; x2]
% Constants for backtracking
beta = 0.5;
gamma = 0.5;
alpha_k = 1;
% Maximum iterations
max_iterations = 100;
iteration_counter = 1;
% Perform Newton's backtracking algorithm
while iteration_counter <= max_iterations
   % Calculate gradient and Hessian at current x_k
   grad_val = vpa(subs(grad_f, [x1, x2], x_k.'), 200);
   hessian_val = vpa(subs(hessian_f, [x1, x2], x_k.'), 200);

   % Calculate d_k
   d_k = vpa(-inv(hessian_val)*grad_val, 200);

   if norm(d_k) <= 10e-3
       break;
   end

   % Calculate f(x_k)
   f_xk = vpa(subs(f, [x1, x2], x_k.'), 200);

   % Compute alpha_k using backtracking line search
   while true
       % Compute x_k_plus_1 and f(x_k_plus_1)
       x_k_plus_1 = x_k + alpha_k*d_k;
       f_xk_plus_1 = vpa(subs(f, [x1, x2], x_k_plus_1.'), 200);

       % Compute RHS of backtracking condition
       rhs_backtracking = vpa(gamma*alpha_k*grad_val.'*d_k, 200);

       % Check backtracking condition
       if f_xk - f_xk_plus_1 >= -rhs_backtracking
           break; % Exit backtracking loop
       else
           alpha_k = beta*alpha_k; % Update alpha_k
       end
   end

   % Update x_k for the next iteration
   x_k = vpa(x_k_plus_1, 200);
```

```
    % Increment iteration counter
    iteration_counter = iteration_counter + 1;
end
disp(['Number of iterations for convergence: ', num2str(iteration_counter)]);
disp(['Final solution: x = [', char(x_k(1)), ', ', char(x_k(2)), ']']);
```

Q2 (d) - Results:

```
>> hw3_q2d
Number of iterations for convergence: 2
Final solution: x = [0, 0]
```

Q2 (e1) - Results:

```
Inf

Hessian matrix is nearly singular. Terminating iteration.
Number of iterations completed:
     4
```

Q2 (e2) - Results:

```
Command Window
    14

2.00000000000000000000000000045

Number of iterations for convergence: 14
Final solution: x = [0.000000000000021245581032739186301357119940668, 0.000000000000021245581032739186301357119940668]
```

Q2 (f) - Code:

```
% Define symbolic variables
syms x1 x2
% Define the function f
f = sqrt(x1^2+1) + sqrt(x2^2+1);
% Calculate gradient of f
grad_f = gradient(f, [x1, x2]);
% Calculate Hessian of f
hessian_f = hessian(f, [x1, x2]);
% Initial guess
```

```
x_k = [10; 10]; % Column vector [x1; x2]
% Constants for backtracking
beta = 0.5;
gamma = 0.5;
alpha_k = 1;
% Maximum iterations
max_iterations = 1000;
iteration_counter = 1;
% Perform Newton's backtracking algorithm
while iteration_counter <= max_iterations
   % Calculate gradient and Hessian at current x_k
   grad_val = vpa(subs(grad_f, [x1, x2], x_k.'), 200);
   hessian_val = vpa(subs(hessian_f, [x1, x2], x_k.'), 200);

   % Calculate d_k
   d_k = vpa(-inv(hessian_val)*grad_val, 200);

   % Calculate f(x_k)
   f_xk = vpa(subs(f, [x1, x2], x_k.'), 200);

   % Compute alpha_k using backtracking line search
   while true
       % Compute x_k_plus_1 and f(x_k_plus_1)
       x_k_plus_1 = x_k + alpha_k*d_k;
       f_xk_plus_1 = vpa(subs(f, [x1, x2], x_k_plus_1.'), 200);

       % Compute RHS of backtracking condition
       rhs_backtracking = vpa(gamma*alpha_k*grad_val.'*d_k, 200);

       % Check backtracking condition
       if f_xk - f_xk_plus_1 >= -rhs_backtracking
           break; % Exit backtracking loop
       else
           alpha_k = beta*alpha_k; % Update alpha_k
       end
   end

   % Update x_k for the next iteration
   x_k = vpa(x_k_plus_1, 200);

   % Increment iteration counter
   iteration_counter = iteration_counter + 1;

   if norm(d_k) <= 10e-3
       break;
   end
    disp(iteration_counter);
   disp(vpa(subs(f, [x1, x2], x_k.'), 30))
end
```

```
disp(['Number of iterations for convergence: ', num2str(iteration_counter)]);
disp(['Final solution: x = [', char(x_k(1)), ', ', char(x_k(2)), ']']);
```

Q2 (f) - Results:

```
Command Window
    620

  2.000049321998121773408839778l9

  Number of iterations for convergence: 621
  Final solution: x = [0.0069681355657111524674063532415594773854162537785395828370611157061954404416616396065023403307812018236
```

The backtracking algorithm achieves convergence in 621 iterations.