
Table of Contents

.....	1
Q2d-i	1
Q2d-ii	1
Q2d-iii	2
Q2d-iiii	3
Q2d-v	4
Plotting	5
Q2d Comments	6

```
clear all
close all
```

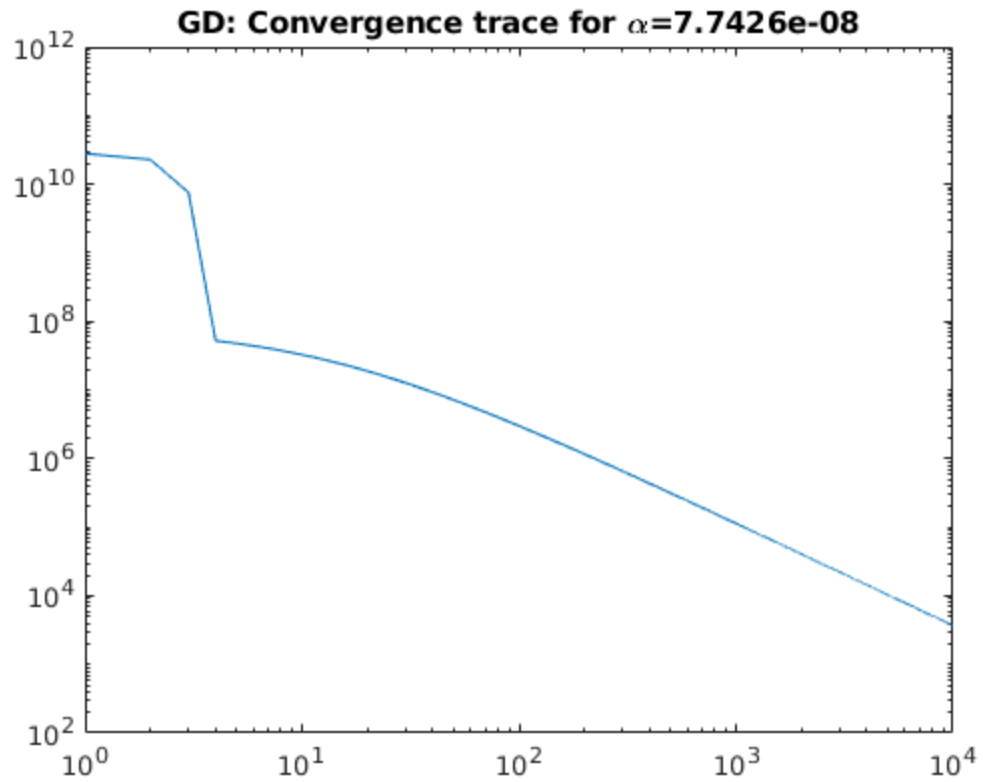
Q2d-i

```
n = 2;
m = 5;
randn('seed', 317) ;
A=randn(n,m); % Locations of m=5 sensors
x=randn(n,1); % True location of x
d=sqrt(sum((A-x*ones(1,m)).^2))+0.05*randn(1,m); % Noise of m=5
sensors
d=d'; % Noise in measurements
x0 = [1000, -500]';

f = @(y) f_func(y, A, d);
df = @(y) df_func(y, A, d);
max_iterations = 1e4;
epsilon = 1e-2;
alpha_range = logspace(1, -10, 100);
```

Q2d-ii

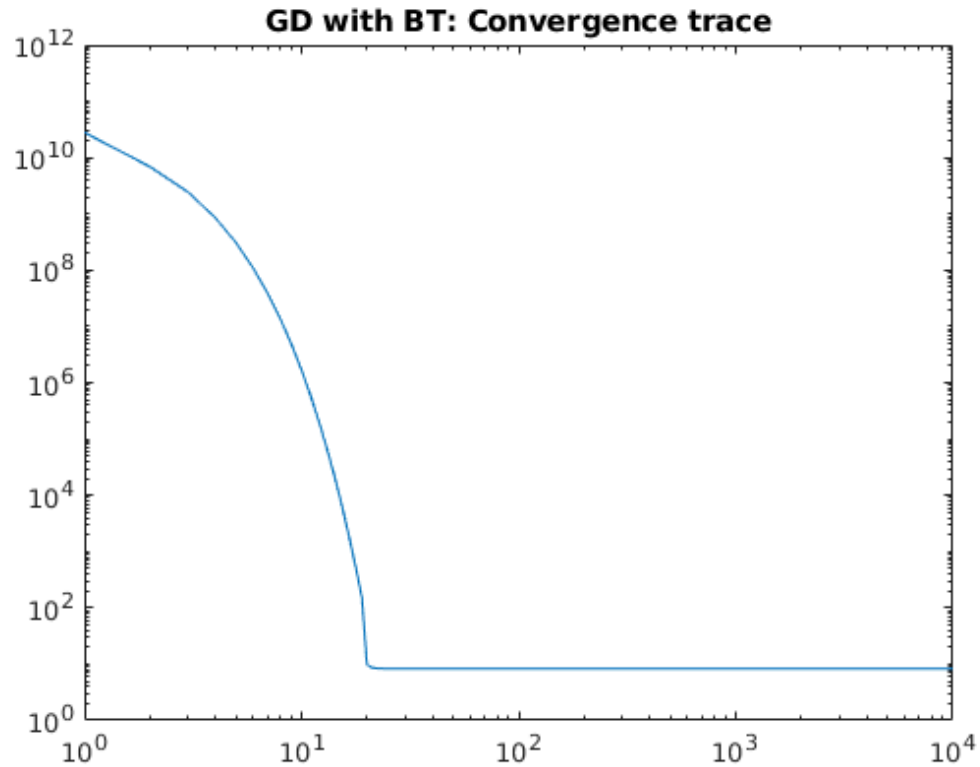
```
for i = 1:length(alpha_range)
    alpha = alpha_range(i);
    [x_trace_gd, trace_gd, status] = gd(df, x0, alpha, max_iterations,
    epsilon);
    if status >= 0
        break
    end
end
figure(1)
loglog(trace_gd)
title("GD: Convergence trace for \alpha=" + num2str(alpha))
```



Q2d-iii

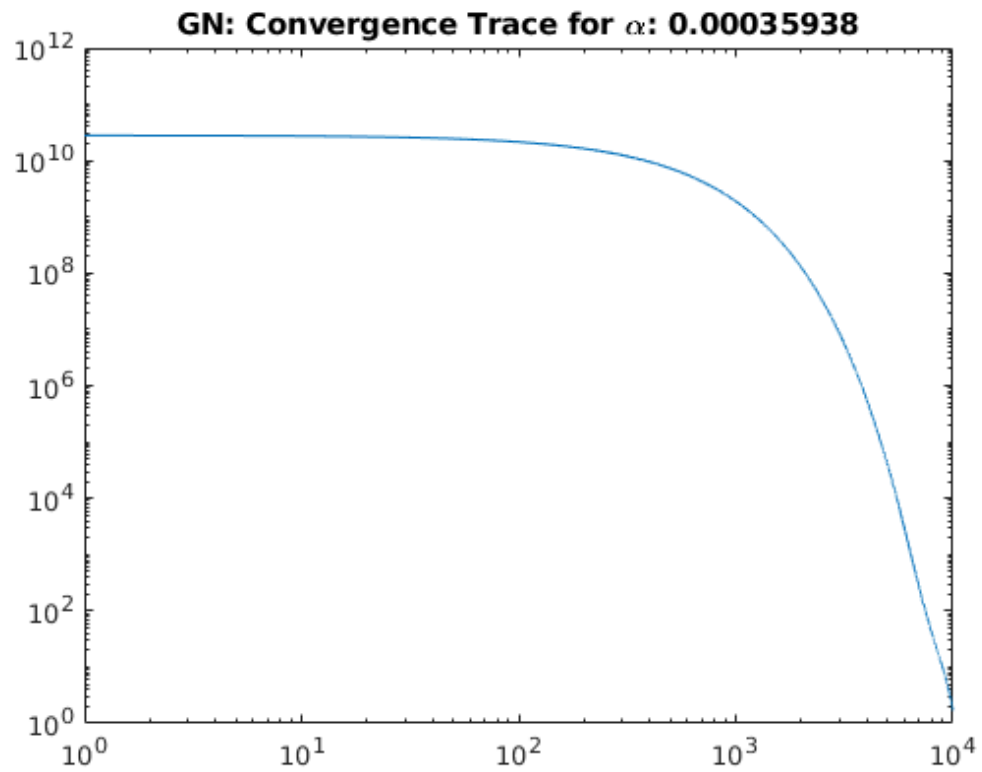
```
[x_trace_gd_bt, trace_gd_bt, status] = gd_bt(f, df, x0, 1, 0.5, 0.5,
max_iterations, epsilon);
figure(2)
loglog(trace_gd_bt)
title("GD with BT: Convergence trace");

r = @(x) norm(x-A)^2 - d.^2;
J = @(x) (2*(x - A))';
```



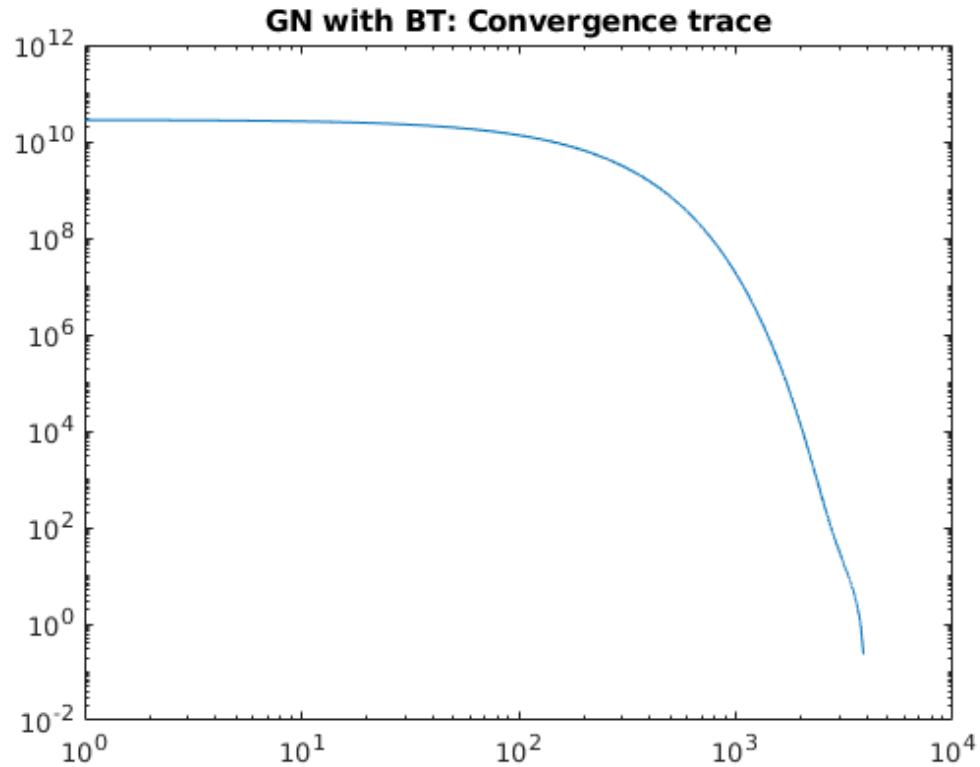
Q2d-iiii

```
warning('off', 'MATLAB:singularMatrix');
warning('off', 'MATLAB:nearlySingularMatrix');
for i = 1:length(alpha_range)
    alpha = alpha_range(i);
    [x_trace_gn, trace_gn, status] = gn(J, r, df, x0, alpha,
    max_iterations, epsilon);
    if status >= 0
        break
    end
end
warning('on', 'MATLAB:singularMatrix');
warning('on', 'MATLAB:nearlySingularMatrix');
figure(3)
loglog(trace_gn)
title("GN: Convergence Trace for \alpha: " + num2str(alpha));
```



Q2d-v

```
[x_trace_gn_bt, trace_gn_bt, status] = gn_bt(J, r, f, df, x0, 1, 0.5,  
0.5, max_iterations, epsilon);  
figure(4)  
loglog(trace_gn_bt)  
title("GN with BT: Convergence trace");
```

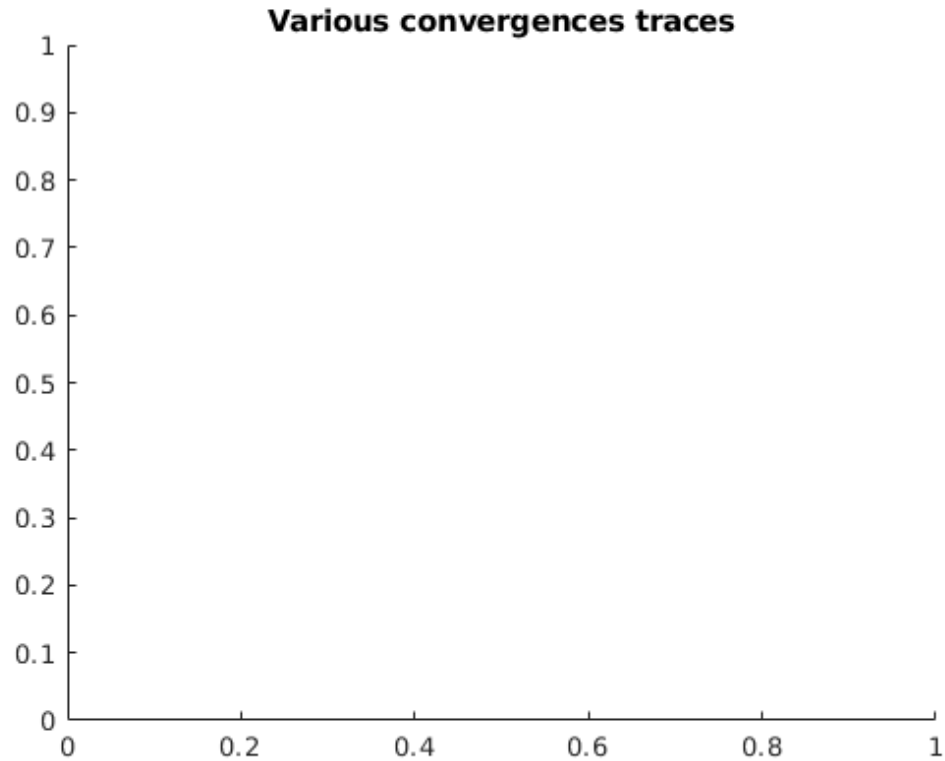


Plotting

```
figure(5)
title("Various convergences traces")
k = 100;

hold on
loglog(f_apply(x_trace_gd(1:k, :),
    f), 'Color', 'k', "DisplayName", "GD")
loglog(f_apply(x_trace_gd_bt(1:k, :),
    f), 'Color', 'm', "DisplayName", "GD with BTLS")
loglog(f_apply(x_trace_gn(1:k, :), f), 'Color', 'b', "DisplayName", "GN")
loglog(f_apply(x_trace_gn_bt(1:k, :), f), 'Color', 'r', "DisplayName", "GN
    with BTLS")
hold off
legend

disp("GD err:" + num2str(norm(x_trace_gd(end) - x)))
disp("GD with Backtracking err:" + num2str(norm(x_trace_gd_bt(end) -
    x)))
disp("GN err:" + num2str(norm(x_trace_gn(end) - x)))
disp("GN with Backtracking err:" + num2str(norm(x_trace_gn_bt(end) -
    x)))
```



Q2d Comments

```
%{
The main tweaks to the algorithms that I found helpful is the early
stopping conditions that forced the function to return when it becomes
clear that the function is diverging. This snippet of code allowed me
to
iterate through various step sizes much faster:
```

```
    elseif isnan(norm(xk)) || ~isfinite(norm(xk))
        status = -1;
        return
    end
```

I also have a similar condition, which I'm not completely sure is correct or not, as there have been cases where the function appeared to be diverged but eventually began to converge (Attached as supplementary picture):

```
    elseif k > 2 && ( trace(k-1) > trace(k-2) )
        status = -1;
        return
```

I obtained a much smaller error using this condition in my GN algorithm

instead of the explicit NAN and inf checks, so I used it instead.

Overall the GN algorithms did a much better job at locating the source in terms of error but the gradient descent quickly seemed to find minimums as demonstrated by figure(5). Perhaps because they descend down where the gradient is most negative.

1: Gradient Descent

The constant step-size gradient takes forever to converge and since I don't have much computing power on my laptop I chose to cap iterations at

1e5 iterations. Although it appears that it is still descending with the

first alpha picked as shown in figure(1).

2: Gradient Descent With Backtracking

Figure(2) shows that this algorithm converged in about 20 iterations which

is great but the error was still significant

3: Damped Gauss-Newton

Although this algorithm took awhile to converge it managed to minimize the

error a significant amount compared the the first 2 algorithms, perhaps if

ran more iterations it would have converged completely

4: Damp Gauss-Newton With Backtracking

This is the only algorithm that converged to a solution and seemed to locate the original source to a very close degree. I suppose it is the most

effective algorithm in this scenario even though it was the most complex.

%}

```
function y = f_apply(X, f)
    y = zeros(length(X), 1);
    for i = 1:length(y)
        y(i) = f(X(i, :));
    end
end
```

```
function f_acc = f_func(x, C, d)
m = length(C);
f_acc = 0;
for i = 1:m
    f_acc = f_acc + (norm(x - C(:,i))^2 - d(i))^2;
end
end
```

```
function df_vec = df_func(x, C, d)
m = length(C);
n = length(x);
df_vec = zeros(n, 1);
for j = 1:n
```

```

        df_acc = 0;
        for i = 1:m
            df_acc = df_acc + 4*(norm(x-C(:,i))^2 - d(i)^2)*(x(j) -
C(j,i));
        end
        df_vec(j) = df_acc;
    end
end

function [x_trace,trace, status] = gd(g, x0, alpha, max_iters,
    epsilon)
trace = zeros(max_iters, 1);
x_trace = zeros(max_iters, length(x0));
xk = x0;
for k = 1:max_iters
    trace(k) = norm(g(xk));
    x_trace(k, :) = xk;
    xk = xk - alpha*g(xk);
    if norm(g(xk)) < epsilon
        status = 1;
        x_trace = x_trace(1:k, :);
        trace = trace(1:k);
        return
    elseif isnan(norm(xk)) || ~isfinite(norm(xk))
        status = -1;
        return
    end
end
status = 0;
end

function [x_trace,trace, status] = gd_bt(f, g, x0, s, alpha, beta,
    max_iters, epsilon)
trace = zeros(max_iters, 1);
x_trace = zeros(max_iters, length(x0));
xk = x0;

for k = 1:max_iters
    trace(k) = norm(g(xk));
    x_trace(k, :) = xk;
    % Determine new step size
    dk = -g(xk); % Negative gradient is descent direction
    i = 0;
    tk = s;
    while f(xk) - f(xk + tk*dk) < -alpha*tk*g(xk)'*dk
        i = i + 1;
        tk = s*beta^i;
    end

    xk = xk + tk*dk;
    % Early stopping conditions
    if norm(g(xk)) < epsilon
        x_trace = x_trace(1:k, :);
        trace = trace(1:k);

```

```

        status = 1;
        return
    elseif isnan(norm(xk)) || ~isfinite(norm(xk))
        status = -1;
        return
    end
end
status = 0;
end

function [x_trace,trace, status] = gn(J_f, r_f, g, x0, alpha,
    max_iters, epsilon)
trace = zeros(max_iters, 1);
x_trace = zeros(max_iters, length(x0));
xk = x0;
for k = 1:max_iters
    trace(k) = norm(g(xk));
    x_trace(k, :) = xk;
    % Early stopping conditions
    if norm(g(xk)) < epsilon
        status = 1;
        x_trace = x_trace(1:k, :);
        trace = trace(1:k);
        return
    elseif k > 2 && ( trace(k-1) > trace(k-2) )
        status = -1;
        return
    elseif isnan(norm(xk)) || ~isfinite(norm(xk))
        status = -1;
        return
    end

    % Damped Gauss Newton
    Jk = J_f(xk);
    rk = r_f(xk);
    dk = (Jk'*Jk)\(Jk'*rk);
    xk = xk - alpha*dk;
end
status = 0;
end

function [x_trace,trace, status] = gn_bt(J_f, r_f, f, g, x0, s, alpha,
    beta, max_iters, epsilon)
trace = zeros(max_iters, 1);
x_trace = zeros(max_iters, length(x0));
xk = x0;
for k = 1:max_iters
    trace(k) = norm(g(xk));
    x_trace(k, :) = xk;
    % Early stopping conditions
    if norm(g(xk)) < epsilon
        status = 1;
        trace = trace(1:k);
        x_trace = x_trace(1:k, :);

```

```

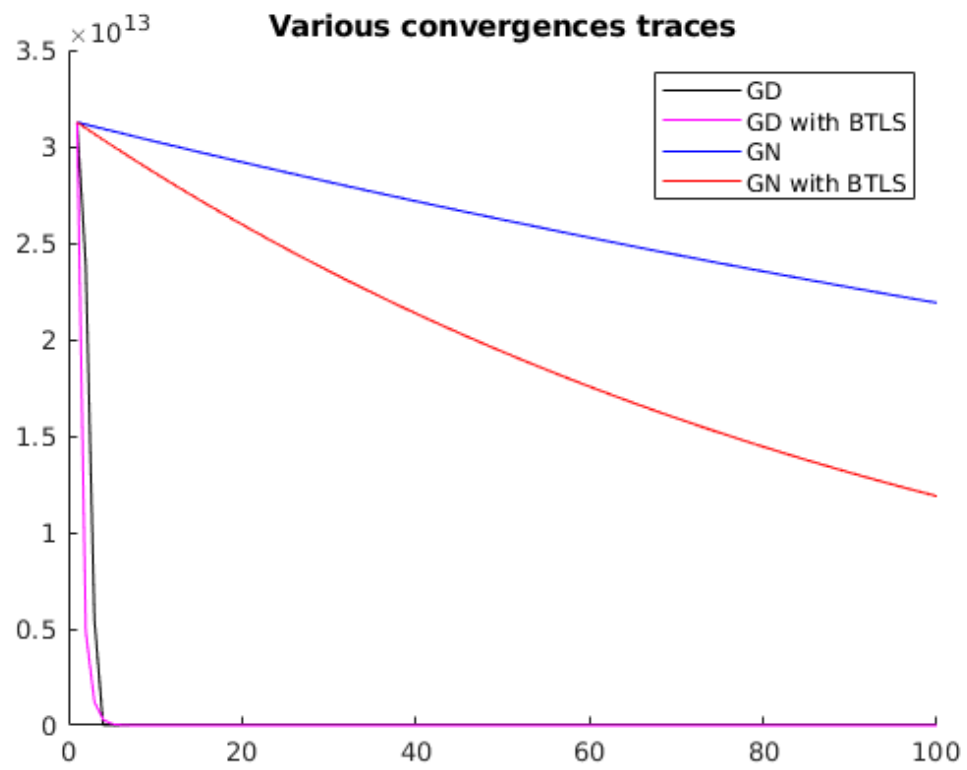
        return
    elseif k > 2 && ( norm(trace(k-1)) > norm(trace(k-2)) )
        status = -1;
        return
    elseif isnan(norm(xk)) || ~isfinite(norm(xk))
        status = -1;
        return
    end

    % Damped Gauss Newton
    Jk = J_f(xk);
    rk = r_f(xk);
    dk = (Jk'*Jk)\(Jk'*rk);

    % Determine new step size
    i = 0;
    tk = s;
    while i < 10 && f(xk) - f(xk + tk*dk) < -alpha*tk*g(xk)'*dk
        i = i + 1;
        tk = s*beta^i;
    end

    xk = xk - tk*dk;
end
status = 0;
end

GD err:4.3654
GD with Backtracking err:1.3275
GN err:0.48248
GN with Backtracking err:0.44904
```



Published with MATLAB® R2018a