Arash Outadi

35898139 HW4

1a) $$\min \left\| \underbrace{\begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix}}_{\bar{A}} x - \underbrace{\begin{bmatrix} b \\ 0 \end{bmatrix}}_{\bar{b}} \right\|_2^2$$

$$\bar{A}'\bar{A}x = \bar{A}'\bar{b}$$

$$\begin{bmatrix} A^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} A \\ \sqrt{\lambda} I \end{bmatrix} x = \begin{bmatrix} A^T & \sqrt{\lambda} I \end{bmatrix} \begin{bmatrix} b \\ 0 \end{bmatrix}$$

$$(A^T A - \lambda I)x = A^T b$$

$$x = (A^T A + \lambda I)^{-1} A^T b$$

$$\|x\|_2^2 = \|(A^T A + \lambda I)^{-1} A^T b\|_2^2$$

$$\|(Q D Q^T + \lambda Q Q^T)^{-1} A^T b\|^2$$

side note: $(Q A Q^T)^{-1} = Q A^{-1} Q^T$

$$\| \cancel{Q}^{1} (D + \lambda I)^{-1} Q^T A^T b \|^2$$

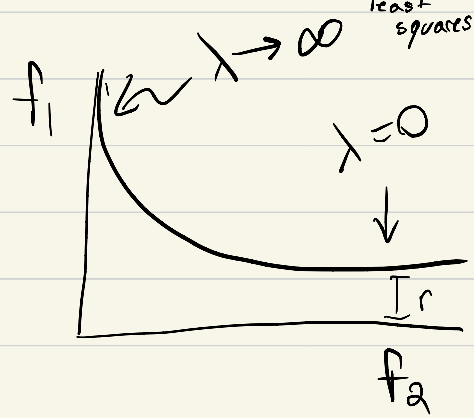
$$\| (D + \lambda I)^{-1} \underbrace{Q^T A^T b}_{g} \|^2$$

$$\boxed{\|x\|_2^2 = \sum_i^n \frac{1}{d_i + \lambda} g_i^2}$$

• As $\lambda \to \infty$, $\|x\|_2^2 = 0$

• As $\lambda = 0$, $\|Ax - b\|_2^2 = r$

left over from least squares



2a) $$\sum_{i=1}^{n} \left( \|x - c_i\|_2^2 - d_i^2 \right)^2 \qquad c_i \in R^n$$

$$\frac{\partial}{\partial x_j} \left( \sum_{i=1}^{n} \left( \sum_{j=1}^{m} (x_j - c_{ij})^2 - d_i^2 \right)^2 \right)$$

$$\sum_{i=1}^{m} \frac{\partial}{\partial x_j} \left( \sum_{j=1}^{m} (x_j - c_{ij})^2 - d_i^2 \right)^2$$

$$\frac{\partial}{\partial x_j} = \sum_{i=1}^{m} 2 \left( \|x - c_i\|_2^2 - d_i^2 \right) \cdot 2 \cdot (x_j - c_{ij})$$

$$\nabla f = \sum_{i=1}^{m} 4 \left( \|x - c_i\|_2^2 - d_i^2 \right) \cdot (x - c_i)$$

Jacobian derivation

2b)

In the form that textbook puts them in

$$\sum_{i=1}^{m} \left( \|x - c_i\|_2^2 - d_i^2 \right)^2 \qquad f_i = \|x - c_i\|_2^2$$

1. $$\sum_{i=1}^{m} \left( f_i - d_i^2 \right) \qquad F = f_i - d_i^2$$

$$\vdots$$

2. $$\min \|F(x)\|^2 \qquad f_m - d_m^2$$

3. $$x_{k+1} = \operatorname{argmin}\left\{ \sum_{i=1}^{m} \left( f_i(x_k) + \nabla f_i(x_k)^T (x - x_k) - d_i^2 \right)^2 \right\}$$

$$r_i(x) = \|x - c_i\|_2^2 - d_i^2$$

$$r_1(x) = \|x - c_1\|_2^2 - d_1^2$$

$$\frac{\partial r}{\partial x_j} = \frac{\partial}{\partial x_j} \sum_{k}^{n} (x_k - c_{ik})^2$$

$$\frac{\partial r}{\partial x_j} = 2(x_j - c_{ij})$$

$$\nabla r_i = 2(x - c_i)$$

LS $\min \|A_k x - b_k\|$

$$A_k = \begin{pmatrix} \nabla f_1(x_k)^T \\ \vdots \\ \nabla f_m(x_k)^T \end{pmatrix} = J(x_k)$$

$$\boxed{J(x) = \begin{bmatrix} \nabla r_1(x)^T \\ \vdots \\ \nabla r_m(x)^T \end{bmatrix}}$$

$$b_k = \begin{pmatrix} \nabla f_1(x_k)^T x_k - f_1(x_k) - d_1^2 \\ \vdots \\ \nabla f_m(x_k)^T x_k - f_m(x_k) - d_m^2 \end{pmatrix}$$

## 2c) Contra-positive proof:

if $A^TA$ is rank deficient then $c_1 \dots c_m$ lie on the same line

$$A = J(x_k) = 2\begin{bmatrix}(x_k - c_1)^T \\ \vdots \\ (x_k - c_m)^T\end{bmatrix}$$

$\text{rank}(A^TA) = \text{rank}(A)$ ~ proof of $N(A) = N(A^TA)$ in past HW

$A \in \mathbb{R}^{m\times 2}$

$\exists y \neq 0$ s.t. $\begin{bmatrix}(x_k - c_1)^T \\ \vdots \\ (x_k - c_m)^T\end{bmatrix} y = 0$ ~ rank deficient

$$\begin{bmatrix}c_1^T \\ \vdots \\ c_m^T\end{bmatrix} y = \begin{bmatrix}x_k \\ \vdots \\ x_k\end{bmatrix} y \Longrightarrow$$

All on same line from def.

$\alpha^T x = \beta \quad \alpha \neq 0 \; \alpha \in \mathbb{R} \quad \beta \in \mathbb{R} \quad x \in \mathbb{R}^n$

in our scenario $\alpha = y \quad \beta = y^T x_k$

# Table of Contents



```
clear all
```

# Q1c

```
load('hw4_1b.mat');
lambdas = [1, 0.01, 10];
k = length(lambdas);

figure(1)
hold on
leg = cell(k+1, 1);
plot(x0);
title('Q1b')
leg{1} = 'x0';
for i = 1:k
    y = lambdas(i);
    cvx_begin quiet
        variable x(n)
        minimize( 0.5*(A*x-b)'*(A*x-b)  + y*norm(x, 1) )
    cvx_end
    plot(x, '--')
    leg{i+1} = strcat('\lambda=', num2str(y));
end
hold off
legend(leg)
```

# Q1d

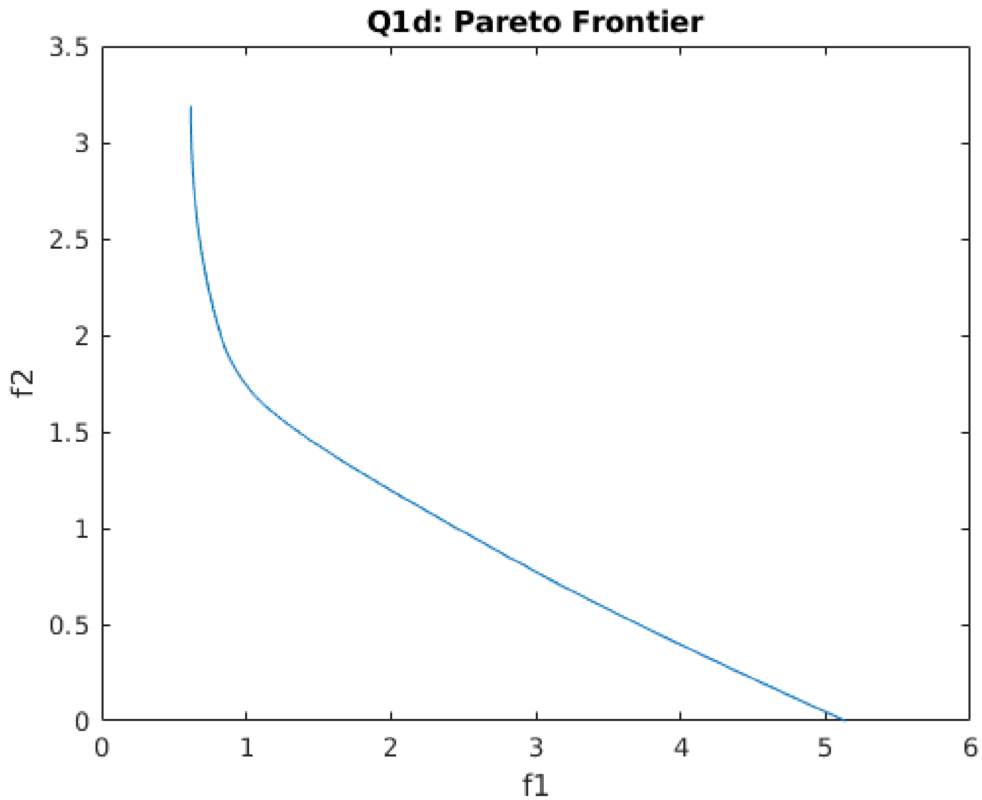
```
lambdas = logspace(-3, 3, 100);
k = length(lambdas);
f1_res = zeros(k, 1);
f2_res = zeros(k, 1);
signal_res = zeros(k, n);

figure(2)
for i = 1:k
    y = lambdas(i);
    cvx_begin quiet
        variable x(n)
        minimize( 0.5*(A*x-b)'*(A*x-b) + y*norm(x, 1) )
    cvx_end
    f1_res(i)= 0.5*norm(A*x - b);
    f2_res(i) = norm(x, 1);
```

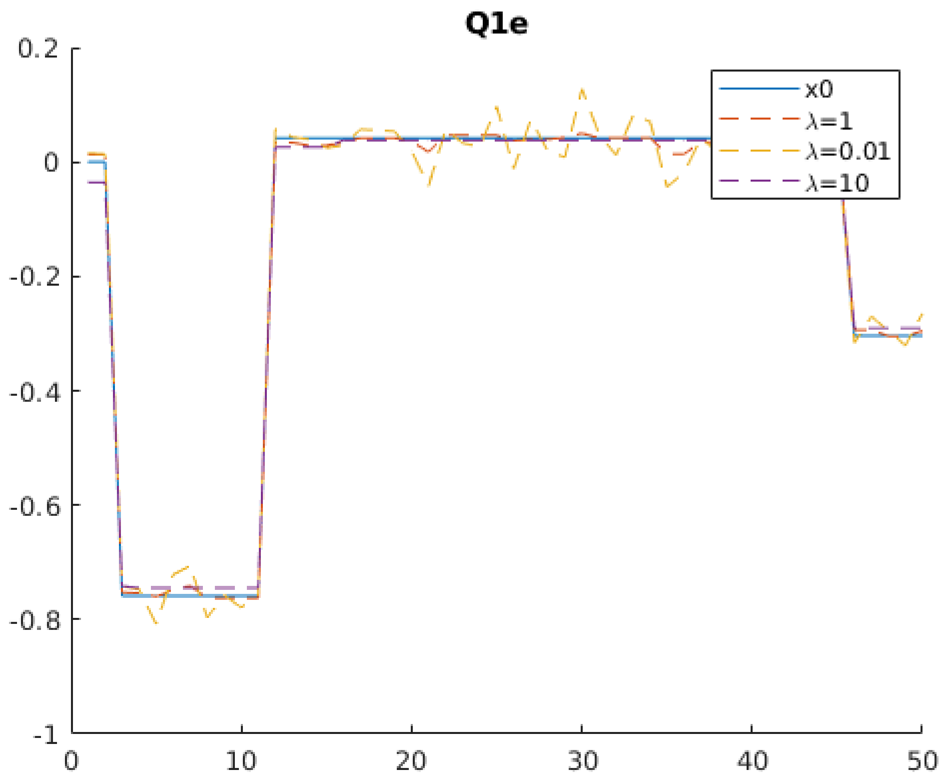
```
    signal_res(i, :) = x;
end
plot(f1_res, f2_res);
title('Q1d: Pareto Frontier')
xlabel('f1')
ylabel('f2')
```

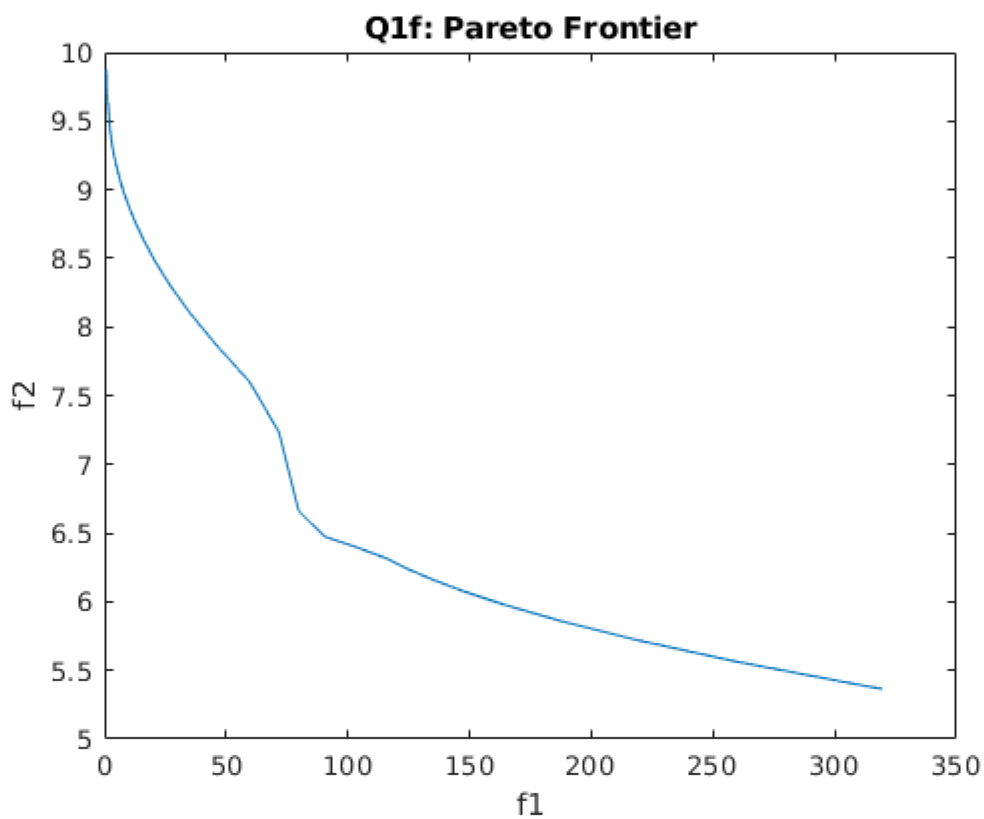
## Q1e

```
load('hw4_1e.mat');

lambdas = [1, 0.01, 10];
k = length(lambdas);

figure(3)
hold on
leg = cell(k+1, 1);
plot(x0);
hold on
title('Q1e')
leg{1} = 'x0';

D = diag(ones(n, 1)) + diag(-ones(n-1, 1), 1);
D = D(1:end-1, :);
for i = 1:k
    y = lambdas(i);
    cvx_begin quiet
        variable x(n)
        minimize( 0.5*(A*x-b)'*(A*x-b)  + y*norm(D*x, 1) )
    cvx_end
    plot(x, '--')
    leg{i+1} = strcat('\lambda=', num2str(y));
end
hold off
legend(leg)

%%Q1f
lambdas = logspace(-3, 3, 100);
k = length(lambdas);
f1_res = zeros(k, 1);
f2_res = zeros(k, 1);
signal_res = zeros(k, n);

figure(4)
for i = 1:k
    y = lambdas(i);
    cvx_begin quiet
        variable x(n)
        minimize( 0.5*(A*x-b)'*(A*x-b)  + y*norm(D*x, 1) )
    cvx_end
    f1_res(i)= 0.5*(A*x-b)'*(A*x-b);
    f2_res(i) = norm(x, 1);
    signal_res(i, :) = x;
```

```
end
plot(f1_res, f2_res);
title('Q1f: Pareto Frontier')
xlabel('f1')
ylabel('f2')
```




Q1e
x0
λ=1
λ=0.01
λ=10
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
10
20
30
40
50


Q1f: Pareto Frontier
f2
f1
10
9.5
9
8.5
8
7.5
7
6.5
6
5.5
5
0
50
100
150
200
250
300
350

*Published with MATLAB® R2018a*

# Table of Contents



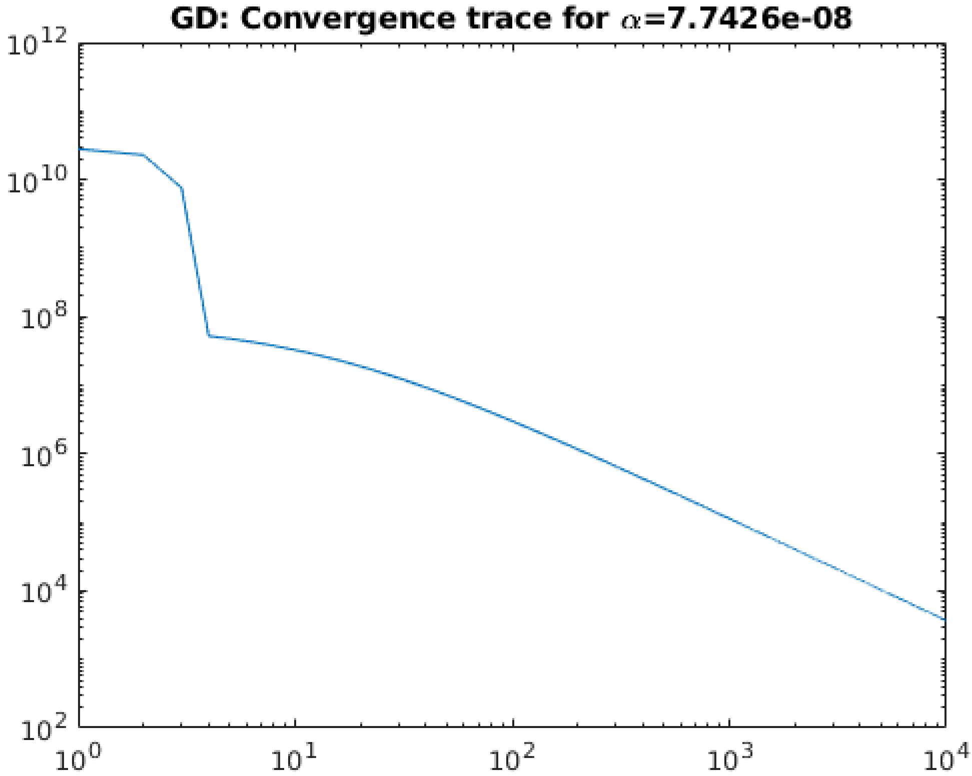
```
clear all
close all
```

# 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);
```

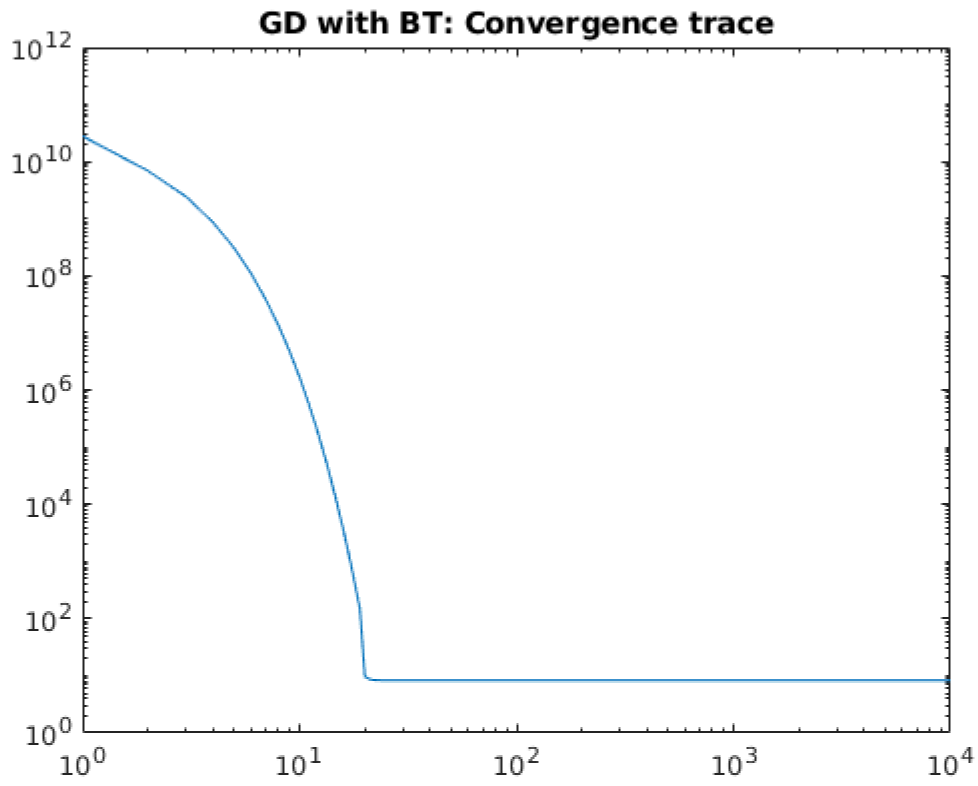
# Q2d-ii

```
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))
```



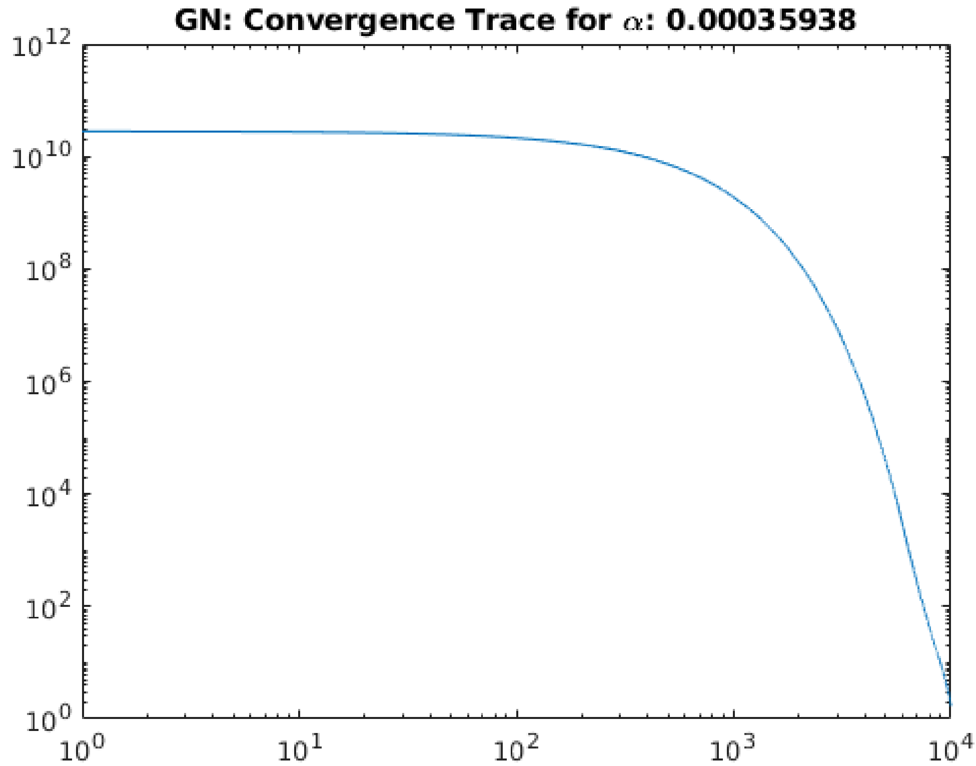

# Q2d-iii

```
[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))';
```



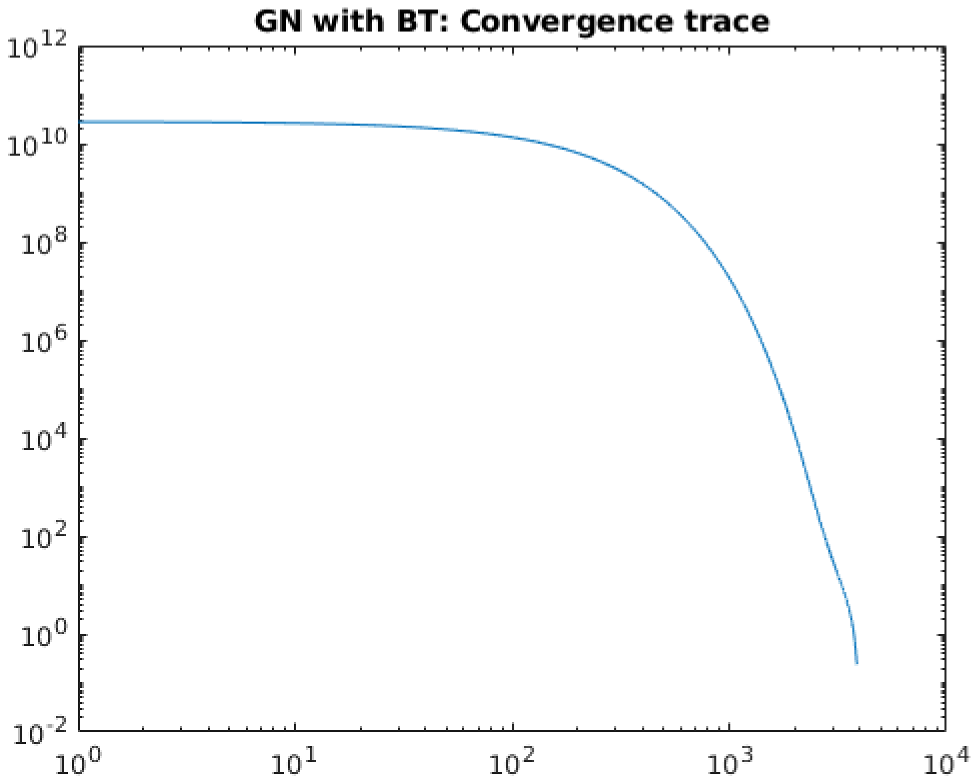

# Q2d-iiii

```
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));
```



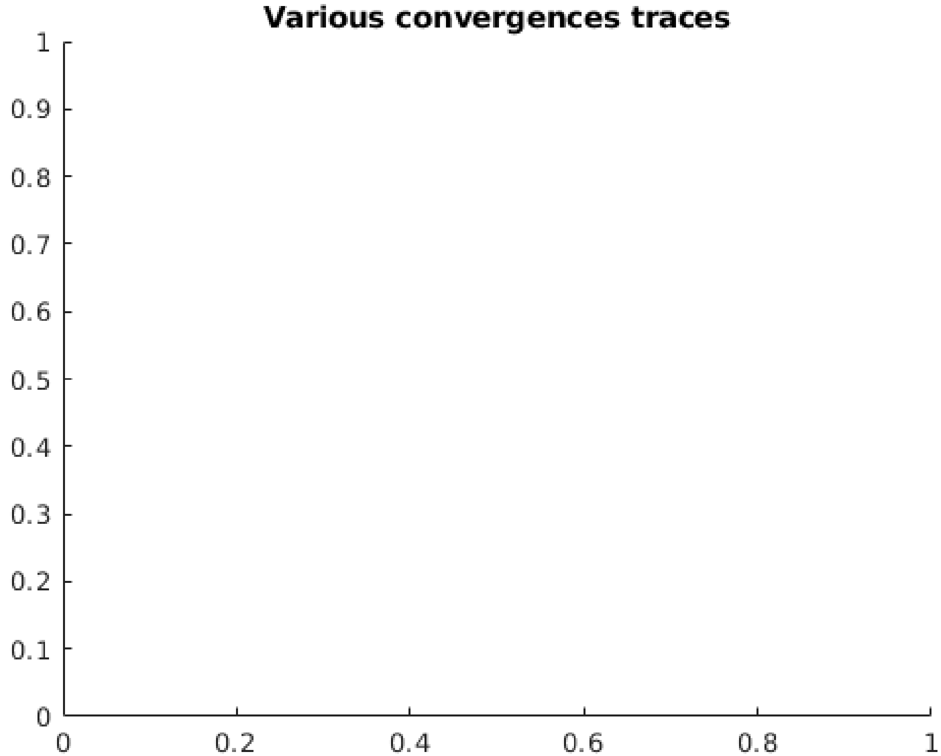

# Q2d-v

```
[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
```

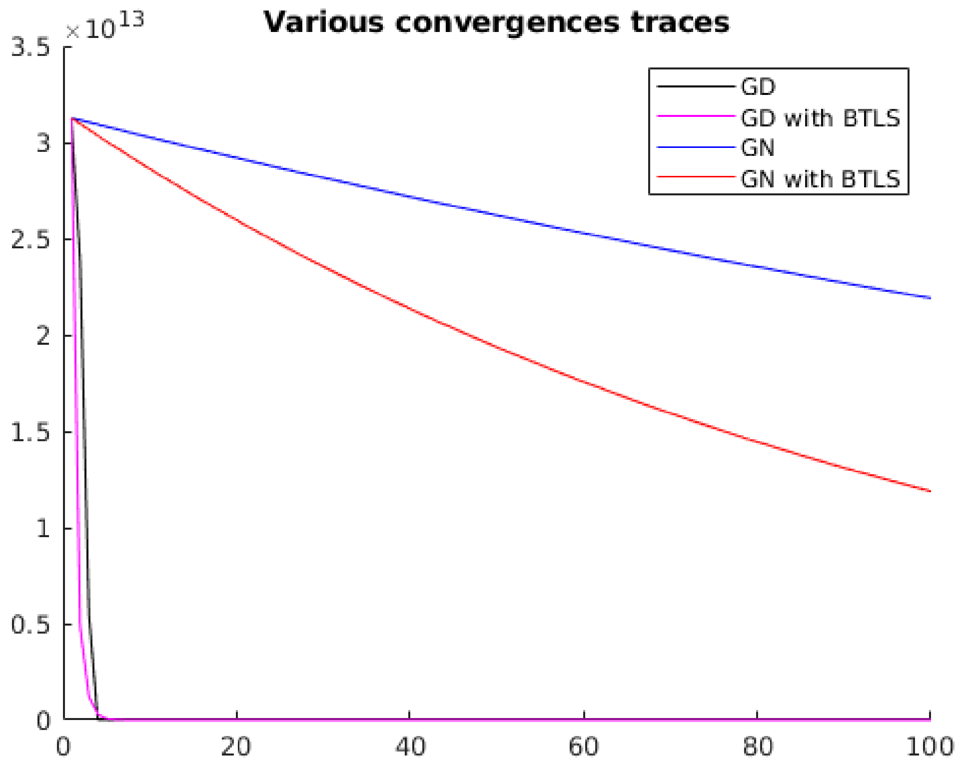
```
instead of the explicit NAN and inf checks, so I used it instead.

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
demostrated 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 signficant
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*