THE UNIVERSITY OF TEXAS
AT AUSTIN

EE381V LARGE SCALE OPTIMIZATION

Problem Set 4

Edited by LaTeX

Department of Computer Science

STUDENT
Jimmy Lin
xl5224

COURSE COORDINATOR
Sujay Sanghavi

UNIQUE NUMBER
17350

RELEASE DATE
October 18, 2014

DUE DATE
October 23, 2014

TIME SPENT
10 hours

October 21, 2014

Table of Contents

List of Figures

Part I
Matlab and Computational Assignment

1 Conjugate Gradient Algorithm

1.1 M_1

Command to be executed in matlab:

```
>> load ConjugateGradient.mat
>> CGS(M1, b1, x)
```

Plot

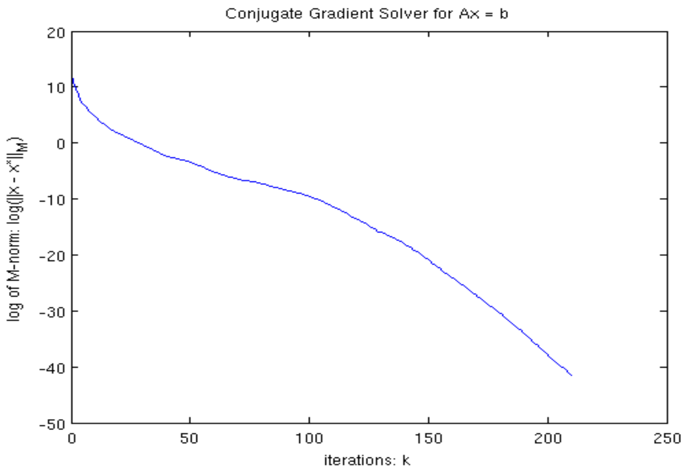

Figure 1: Conjugate Gradient Solver for M_1

1.2 M_2

Command to be executed in matlab:

```
>> load ConjugateGradient.mat
>> CGS(M2, b2, x)
```

Plot

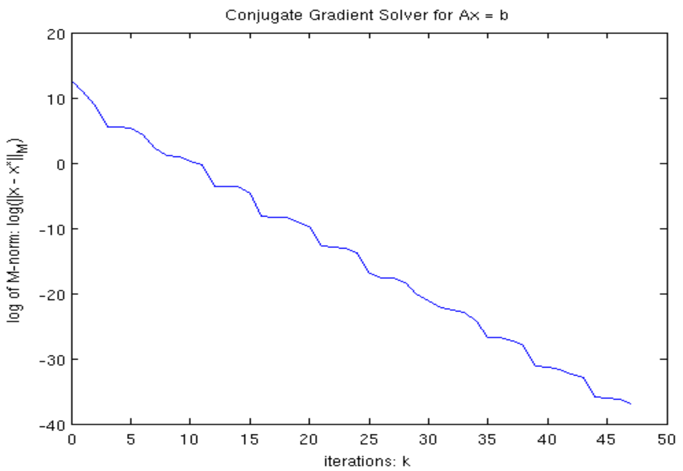

Figure 2: Conjugate Gradient Solver for M_2

2 Newtons Method

3 Central Path

4 Larger Linear Program

5 Gradient, Conjugate Gradient, Newton and BFGS

Part II
Written Problems

A Codes Printout

A.1 Conjugate Gradient Algorithm

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 1. Conjugate Gradient Algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function CGS(M, b, x_opt)
    [R, C] = size(M);
    assert (R == C, 'M should be square matrix.');
    assert (R == size(b,1), 'Dim of M and b should be consistent.');

    EPSILON = 10e-10; % how close solution do we need
    x_0 = zeros(size(x_opt)); % initial point

    listK = [];
    listlogMdiff = [];

    k = 0;
    x = x_0;
    r = b - M * x; % residual
    p = r;
    while 1,
        diff = x - x_opt;
        logMdiff = log(diff' * M * diff);

        fprintf ('iteration: %d, log(||x - x*||_M) = %f \n', k, logMdiff)
        listK = [listK k];
        listlogMdiff = [listlogMdiff logMdiff];

        alpha = (r' * r) / (p' * M * p);
        x = x + alpha * p;
        r_new = r - alpha * M * p;
        if r_new < EPSILON,
            break
        end
        beta = (r_new' * r_new) / (r' * r);
        p = r_new + beta * p;
        r = r_new;
        k = k + 1;
    end
    plot (listK, listlogMdiff)
    title('Conjugate Gradient Solver for Ax = b')
    xlabel('iterations: k')
    ylabel('log of M-norm: log(||x - x*||_M)')
end
```

A.2 Newtons Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 2. Newton Method
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function Newton(m)
    EPSILON = 10e-10; % how close solution do we need
    x_0 = zeros(size(x_opt)); % initial point
    t = 1; % step size fixed at 1
    x_opt = 0;

    listK = [];
    listlogMdiff = [];

    k = 0;
    x = x_0;
    while 1,
        diff = x - x_opt;
        logMdiff = log(diff' * M * diff);

        fprintf ('iteration: %d, log(||x - x*||_M) = %f \n', k, logMdiff)
        listK = [listK k];
        listlogMdiff = [listlogMdiff logMdiff];

        grad = grad_func(x, m);
        hess = hess_func(x, m);
        x = x + t * grad' * inv(hess) * grad;

        x
        k = k + 1;
    end
    plot (listK, listlogMdiff)
    title('Conjugate Gradient Solver for Ax = b')
    xlabel('iterations: k')
    ylabel('log of M-norm: log(||x - x*||_M)')
end

function val = func(x, m)
    nornx = norm(x, 2);
    val = normx^3 + 0.5 * m * normx^2;
end

function val = grad_func(x, m)

end

function val = hess_func(x, m)

end
```

A.3 Central Path

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% 3. Central Path
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [x, k, logNorm2] = CP(M, b)
    k = [];
    logNorm2 = [];


end
```

A.4 Larger Linear Program

A.5 Gradient and Newton