HW8

Carl Mueller
CSCI 5254 - Convex Optimization

May 2, 2018

8.16

a)

Gradient Method

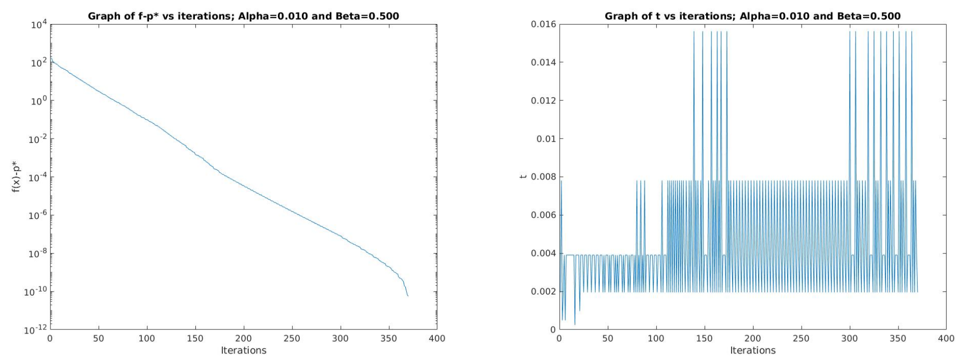

Figure 1: ALPHA=.01, BETA=.5

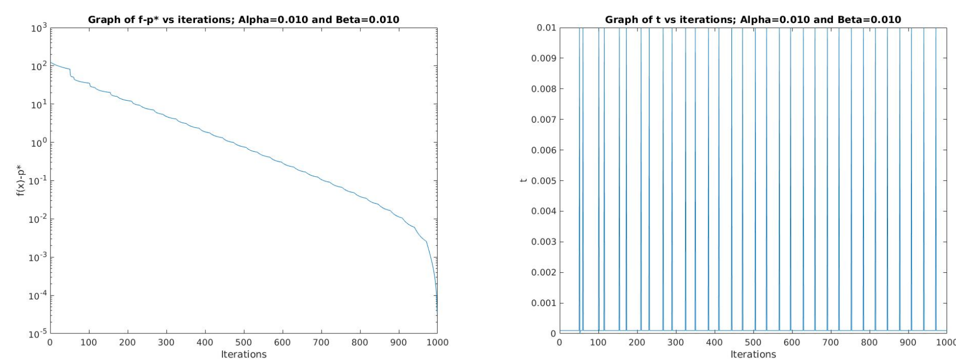

Figure 2: ALPHA=.01, BETA=.01

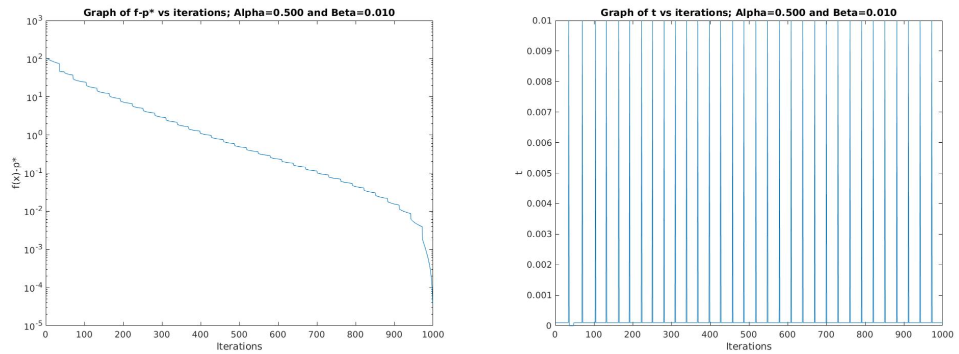

Figure 3: ALPHA=.50, BETA=.01

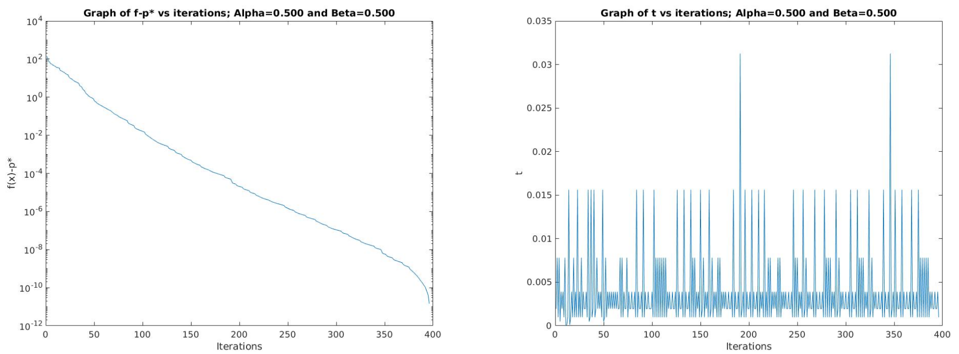

Figure 4: ALPHA=.50, BETA=.50

```
iter = 1000;
nu = .0001;
beta = .01;
alpha = .5;
n = 100;
m = 200;
x = zeros(n, 1);
A = randn(m,n);

V = []
I = []
T = []
for i = 1:iter
    % function evaluation
    f = -sum(log(1-A*x))-sum(log(1+x)) - sum(log(1-x));
    % gradient
    grad = A'*(1./(1-A*x)) - 1./(1+x) + 1./(1-x);
    % breaking criterion using nu as threshold
```

```
    if norm(grad) < nu
        break
    end
    % Gradient direction.
    dir = -grad;
    % second compoenent of backtracking
    fprime = grad'*dir;
    t = 1;
    while ((max(A*(x+t*dir)) >= 1) || (max(abs(x+t*dir)) >= 1))
        t = beta*t;
    end
    % backtracking algorithm
    while ( -sum(log(1-A*(x+t*dir))) - sum(log(1-(x+t*dir).^2)) > f + alpha*t
        t = beta*t;
    end
    % update step
    x = x+t*dir;
    T = [T; t]
    V = [V; f];
    I = [I ; i]
end

f_minus_p = [];
for i = 1:length(V)
    diff = V(i) - f
    f_minus_p = [f_minus_p; diff]
end
f_minus_p
f
figure(1)
plot(I,f_minus_p);
set(gca,'yscale','log');
titlestr = "Graph of f-p* vs iterations; Alpha=%0.3f and Beta=%0.3f";
str = sprintf(titlestr,alpha,beta);
title(str);
xlabel("Iterations");
ylabel("f(x)-p*");

figure(2)
plot(I,T);
titlestr = "Graph of t vs iterations; Alpha=%0.3f and Beta=%0.3f";
str = sprintf(titlestr,alpha,beta);
title(str);
xlabel("Iterations");
ylabel("t");
```

b)

Newton's Method

This approach clearly takes many less iterations and is always terminated based on the quit criteria rather than the max number of iterations.

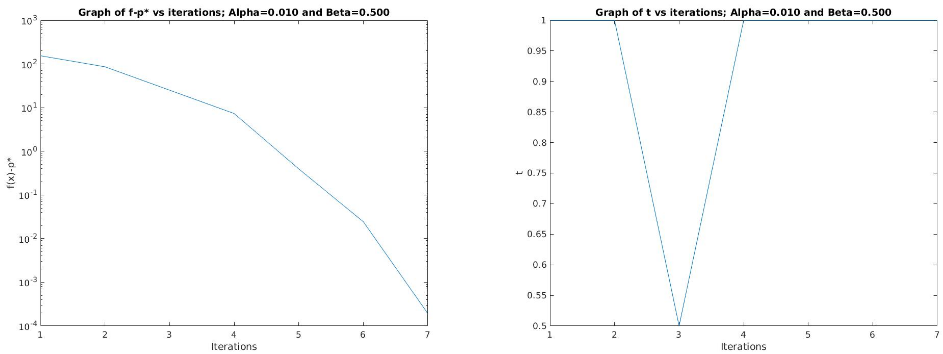

Figure 5: ALPHA=.01, BETA=.5

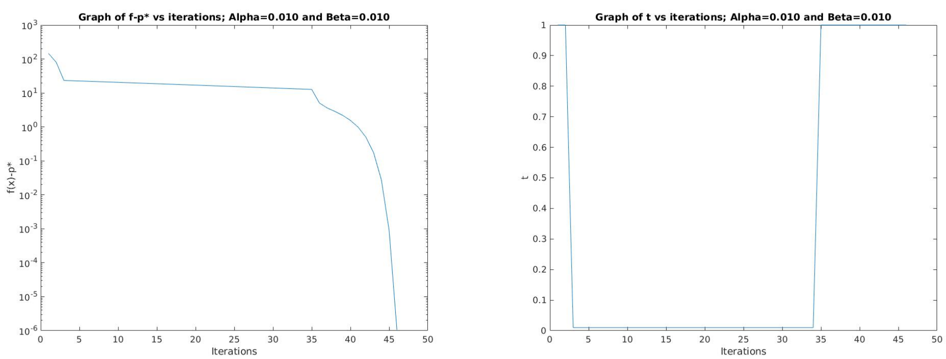

Figure 6: ALPHA=.01, BETA=.01

```
iter = 1000;
nu = .00000001;
beta = .01;
alpha = .50;
n = 100;
m = 200;
x = zeros(n, 1);
A = randn(m,n);


x = zeros(n,1);
```

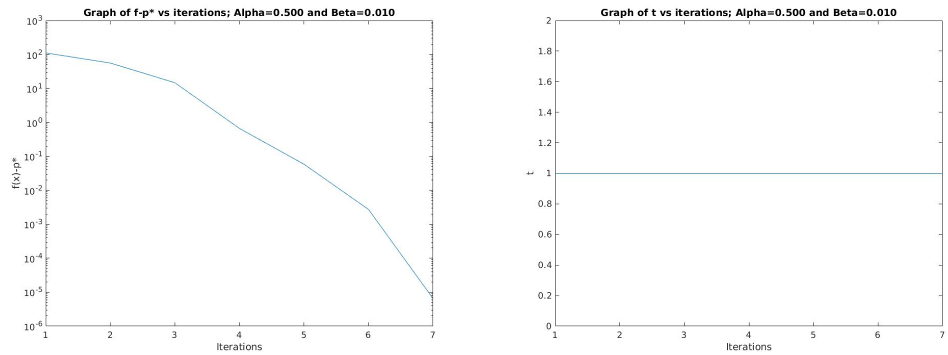

Figure 7: ALPHA=.50, BETA=.01

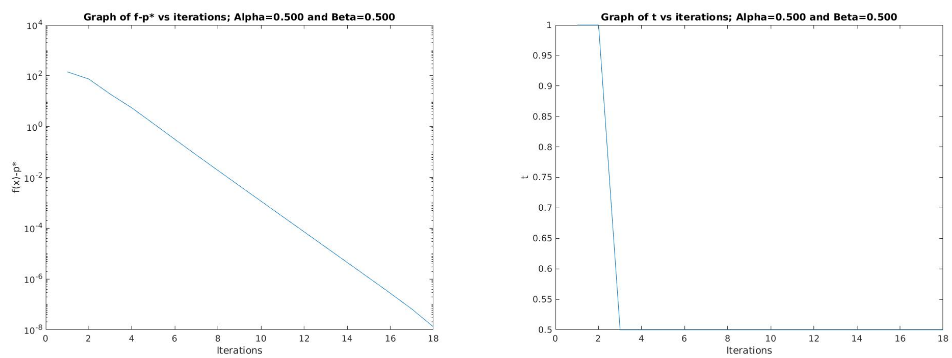

Figure 8: ALPHA=.50, BETA=.50

```
V = []
I = []
T = []
for i = 1:iter
    f = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
    d = 1./(1-A*x);
    % first order derivative
    grad = A'*d - 1./(1+x) + 1./(1-x);
    % second order derivative i.e. hessian
    hessian = A'*diag(d.^2)*A + diag(1./(1+x).^2 + 1./(1-x).^2);
    % direction
    dir = -hessian\grad;
    % lambda^2 i.e decrement
    lambda_2 = grad'*dir;
    t = 1;
    while ((max(A*(x+t*dir)) >= 1) || (max(abs(x+t*dir)) >= 1))
        t = beta*t;
    end
```

```
    % breaking criteria
    if abs(lambda_2) < nu
        break;
    end
    % backtracking algorithm
    while ( -sum(log(1-A*(x+t*dir))) - sum(log(1-(x+t*dir).^2)) > f + alpha*t
        t = beta*t;
    end
    % update step
    x = x+t*dir;
    V = [V; f];
    I = [I ; i];
    T = [T; t];
end

f_minus_p = [];
for i = 1:length(V)
    diff = V(i) - f
    f_minus_p = [f_minus_p; diff]
end
f_minus_p
f
figure(1)
semilogy(I, f_minus_p)
titlestr = "Graph of f-p* vs iterations; Alpha=%0.3f and Beta=%0.3f";
str = sprintf(titlestr,alpha,beta);
title(str);
xlabel("Iterations");
ylabel("f(x)-p*");

figure(2)
plot(I,T);
titlestr = "Graph of t vs iterations; Alpha=%0.3f and Beta=%0.3f";
str = sprintf(titlestr,alpha,beta);
title(str);
xlabel("Iterations");
ylabel("t");
```

9.31

Delayed Hessian Update:

Code:

```
clear all;
```

```
iter = 1000;
nu = .00000001;
beta = .5;
alpha = .01;
n = 100;
m = 200;
x = zeros(n, 1);
A = randn(m,n);

GD = []

step = [1,5,10,20]
for N = 1:length(step)
    V = [];
    I = [];
    T = [];
    for i = 1:iter
        f = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
        d = 1./(1-A*x);
        % first order derivative
        grad = A'*d - 1./(1+x) + 1./(1-x);
        % second order derivative i.e. hessian
        if i == 1 || mod(i, N) == 0
            hessian = A'*diag(d.^2)*A + diag(1./(1+x).^2 + 1./(1-x).^2);
        end
        % direction
        dir = -hessian\grad;
        % lambda^2 i.e decrement
        lambda_2 = grad'*dir;
        t = 1;
        while ((max(A*(x+t*dir)) >= 1) || (max(abs(x+t*dir)) >= 1))
            t = beta*t;
        end
        % breaking criteria
        if abs(lambda_2) < nu
            break;
        end
        % backtracking algorithm
        while ( -sum(log(1-A*(x+t*dir))) - sum(log(1-(x+t*dir).^2)) > f + alp
            t = beta*t;
        end
        % update step
        x = x+t*dir;
        V = [V; f];
```

```
        I = [I ; i];
        T = [T; t];
    end
    f_minus_p = [];
    for i = 1:length(V)
        diff = V(i) - f
        f_minus_p = [f_minus_p; diff]
    end
    GD = [GD ; {f, f_minus_p, V, I, T}]
    x = zeros(n, 1);
end


figure(1)
D = GD(1, :)
semilogy(D{4}, D{2})
hold on
D = GD(2, :)
semilogy(D{4}, D{2})
hold on
D = GD(3, :)
semilogy(D{4}, D{2})
hold on
D = GD(4, :)
semilogy(D{4}, D{2})
hold off
titlestr = "Graph of delayed Hessian Update for Netwon's method";
str = sprintf(titlestr,alpha,beta);
title(str);
xlabel("Iterations");
ylabel("f(x)-p*");
legend("Newton","N=5","N=10","N=20")
```

0.0.1 b)

9.31

Delayed Hessian Update:

Code:

```
...

for i = 1:iter
```

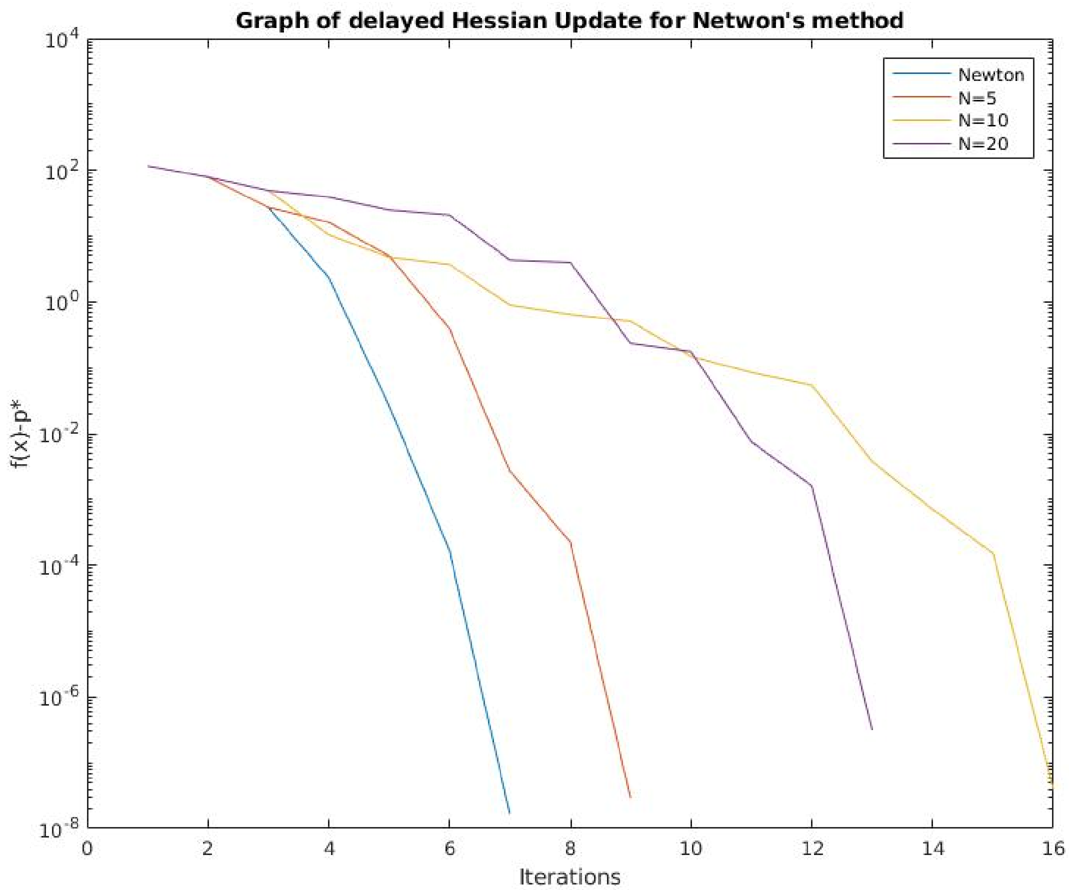

Figure 9: Different iterations counts N until Hessian update.

```
  f = -sum(log(1-A*x)) - sum(log(1+x)) - sum(log(1-x));
  d = 1./(1-A*x);
  % first order derivative
  grad = A'*d - 1./(1+x) + 1./(1-x);
  % diagnal of the second order derivative i.e. hessian
 hessian = diag(diag(A'*diag(d.^2)*A + diag(1./(1+x).^2 + 1./(1-x).^2)));
```

...

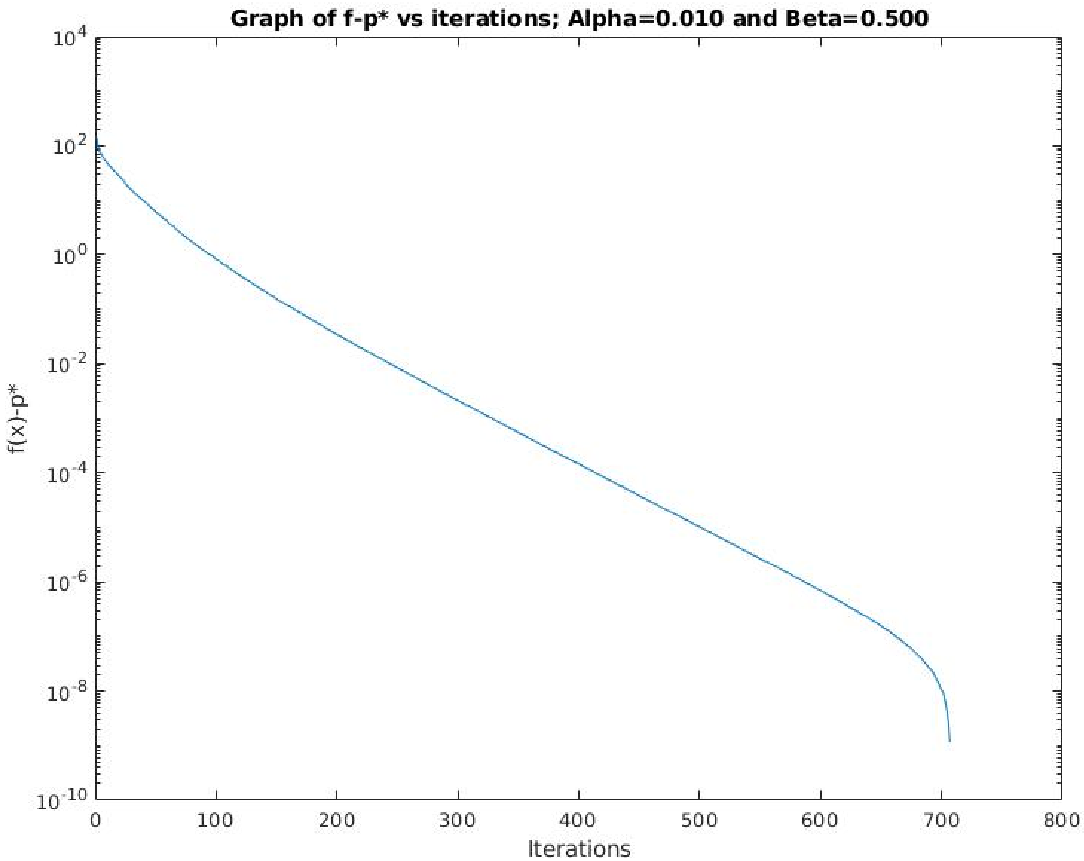

Figure 10: Using the diagonal of the hessian. There is a clear increase in the required number of iterations

11.1

We are given the minimization problem:

$$\begin{aligned} \text{minimize} \quad & x^2 + 1 \\ \text{subject to} \quad & 2 \leq x \leq 4 \end{aligned}$$

We used the log barrier approximation as follows:

$$\hat{I}(x) = -log(x-2) - log(4-x)$$

Code:

```
t = [.01, .02, .04, .08, .16, .32, .64, 1.28, 2.56, 5.12, 10.80]

figure(1)
fplot(@(x) power(2,x)+1, [0,6])
hold on
```

```
for i = 1:length(t)
    fplot(@(x) power(2,x)+1 + (1/t(i))*(-log(x-2)-log(4-x)))
end
hold off
xlabel("x")
ylabel("function value")
title("Log Barrier of f for various t values.")
legend("t=.01","t=.02","t=.04","t=.08","t=.16","t=.32","t=.74","t=1.28","t=5.
```

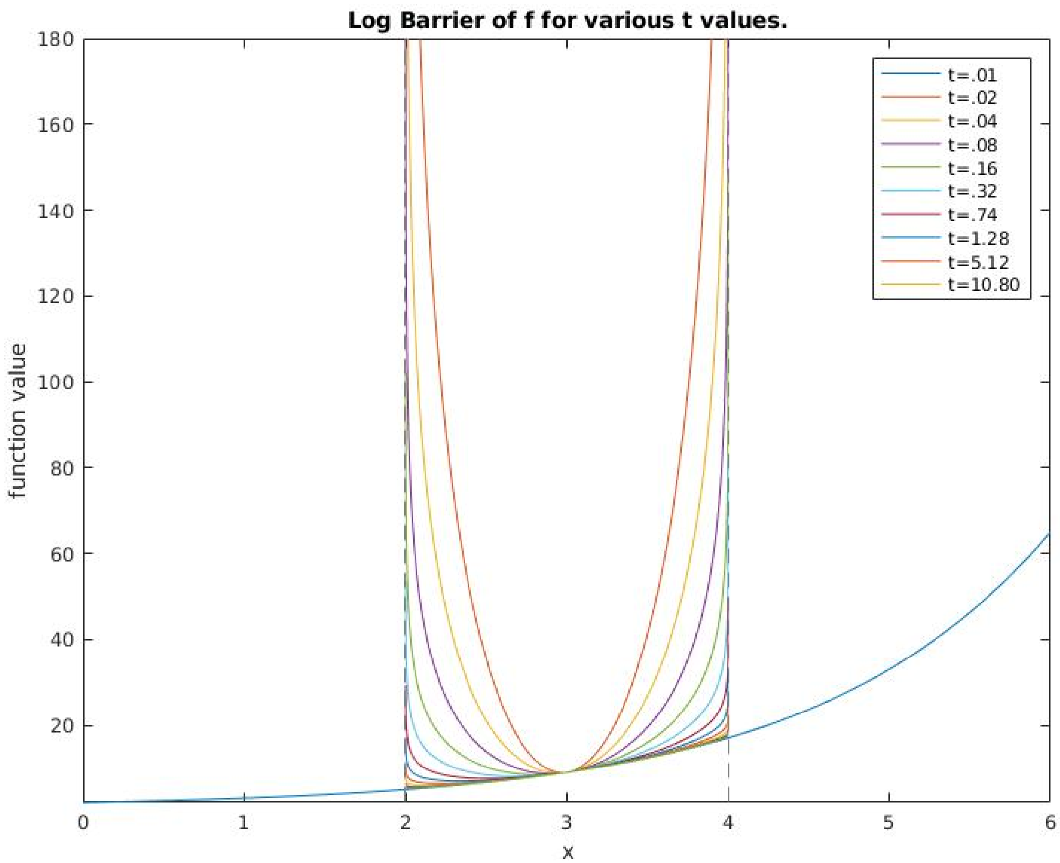

Figure 11: Log Barrier plotting verus f(x) for various t values.

11.22

We use the following optimization problem. We use the bounds of our x values of the constraints using u and l to constrain the log barrier of the rectangle.
Note: $A^+ = max(A, 0)$ and $A^- = max(-A, 0)$

$$\begin{array}{ll} \text{minimize} & -\sum_{i=1}^{n} log(u_i - l_i) \\ \text{subject to} & A^+u - A^-l \preceq b \\ & l \preceq u \end{array}$$

$$\psi = t - \sum_{i=1}^{n} log(u_i - l_i) - \sum_{i=1}^{n} log(b - A^+u_i + A^-l_i)$$

$$\nabla\psi = t \begin{bmatrix} I \\ -I \end{bmatrix} diag(u - l)^{-1}\mathbf{1} + \begin{bmatrix} -A^{-T} \\ A^{+T} \end{bmatrix} diag(b - A^+u + A^-l)^{-1}\mathbf{1}$$

$$\nabla^2\psi = t \begin{bmatrix} I \\ -I \end{bmatrix} diag(u - l)^{-2} \begin{bmatrix} I \\ -I \end{bmatrix}^T + \begin{bmatrix} -A^{-T} \\ A^{+T} \end{bmatrix} diag(b - A^+u + A^-l)^{-2} \begin{bmatrix} -A^{-T} \\ A^{+T} \end{bmatrix}^T$$