EECS 227C - Homework III

Walid Krichene

March 20, 2014

1 Consider the two dimensional optimization problem

$$\text{minimize} \quad \frac{1}{2n}\sum_{k=1}^{n}\left(x_1\cos\frac{k\pi}{n}+x_2\sin\frac{k\pi}{n}\right)^2$$

(a) Prove this is a strongly convex quadratic cost function, and compute the Lipschitz constant of the gradient and strong convexity parameters. Compute the optimal point x_{opt}.

We have

$$f(x)=\frac{1}{2}x^TQx$$

where

$$Q=\frac{1}{n}\sum_{k=1}^{n}\begin{pmatrix}\cos^2\frac{k\pi}{n} & \cos\frac{k\pi}{n}\sin\frac{k\pi}{n}\\ \cos\frac{k\pi}{n}\sin\frac{k\pi}{n} & \sin^2\frac{k\pi}{n}\end{pmatrix}$$

the characteristic polynomial of Q is

$$\begin{aligned}\mathcal{X}(\lambda)&=\det(\lambda I-Q)\\ &=\left(\lambda-\frac{1}{n}\sum_k\cos^2\frac{k\pi}{n}\right)\left(\lambda-\frac{1}{n}\sum_k\sin^2\frac{k\pi}{n}\right)-\frac{1}{n^2}\left(\sum_k\cos\frac{k\pi}{n}\sin\frac{k\pi}{n}\right)^2\\ &=\lambda^2-\frac{1}{n}\left(\sum_k\cos^2\frac{k\pi}{n}+\sum_k\sin^2\frac{k\pi}{n}\right)\lambda+\frac{1}{n^2}\left(\sum_k\cos^2\frac{k\pi}{n}\right)\left(\sum_k\sin^2\frac{k\pi}{n}\right)-\frac{1}{n^2}\left(\sum_k\cos\frac{k\pi}{n}\sin\frac{k\pi}{n}\right)^2\\ &=\lambda^2-\lambda+a\end{aligned}$$

where

$$\begin{aligned}a&=\frac{1}{n^2}\left(\sum_k\cos^2\frac{k\pi}{n}\right)\left(\sum_k\sin^2\frac{k\pi}{n}\right)-\frac{1}{n^2}\left(\sum_k\cos\frac{k\pi}{n}\sin\frac{k\pi}{n}\right)^2\\ &=\frac{1}{n^2}\left(\frac{1}{2}\sum_k1+\cos2\frac{k\pi}{n}\right)\left(\frac{1}{2}\sum_k1-\cos2\frac{k\pi}{n}\right)-\frac{1}{n^2}\left(\sum_k\sin2\frac{k\pi}{n}\right)^2\\ &=\frac{1}{n^2}\left(\frac{n}{2}\right)\left(\frac{n}{2}\right)-0\\ &=\frac{1}{4}\end{aligned}$$

using the fact that $\sum_{k=1}^n\sin2\frac{k\pi}{n}=\sum_{k=1}^n\cos2\frac{k\pi}{n}=0$ (for example, $\sum_{k=1}^n\cos2\frac{k\pi}{n}=\text{Re}(\sum_{k=1}^n e^{i\frac{2k\pi}{n}})=\text{Re}(e^{i\frac{2\pi}{n}}\frac{1-e^{i\frac{2n\pi}{n}}}{1-e^{i\frac{2\pi}{n}}})=0$).

Thus

$$\mathcal{X}(\lambda) = \lambda^2 - \lambda + \frac{1}{4} = (\lambda - \frac{1}{2})^2$$

therefore Q is a symmetric matrix (thus diagonalizable) with eigenvalues $\frac{1}{2}$, i.e. $Q = \frac{1}{2}I$, and $f(x) = \frac{1}{4}\|x\|^2$. It follows that f is strictly convex with parameter $\ell = \frac{1}{4}$, has Lipschitz gradient with Lipschitz constant $\frac{1}{4}$, and the minimizer is $x_{opt} = 0$.

(b) Consider the stochastic gradient method with step size 1 initialized at the point $(0, 1)$. Compute

$$\mathbb{E}\,\|x^n - x_{opt}\|^2$$

when the increments are chosen with replacement.

At step n, if we draw k, then the stochastic gradient is

$$\begin{aligned} g^n &= \nabla_x \frac{1}{2}\left(\cos\frac{k\pi}{n}x_1 + \sin\frac{k\pi}{n}x_2\right)^2 \\ &= \left(\cos\frac{k\pi}{n}x_1 + \sin\frac{k\pi}{n}x_2\right)\begin{pmatrix}\cos\frac{k\pi}{n}\\ \sin\frac{k\pi}{n}\end{pmatrix} \\ &= z_k z_k^T x^n \end{aligned}$$

where $z_k = \begin{pmatrix}\cos\frac{k\pi}{n}\\ \sin\frac{k\pi}{n}\end{pmatrix}$. It is easy to check that

$$\begin{aligned} \mathbb{E}[g^n|x^n] = \mathbb{E}[z_k z_k^T x^n|x^n] &= \left(\frac{1}{n}\sum_k z_k z_k^T\right)x_n \\ &= Qx_n \end{aligned}$$

which is the gradient of $\frac{1}{2}x^TQx$. Now we have

$$x^{n+1} = x^n - g^n = (I - z_k z_k^T)x^n$$

therefore we have

$$\begin{aligned} \mathbb{E}\,\|x^{n+1}\|^2 &= \mathbb{E}[\mathbb{E}[\|x^{n+1}\|^2|x_n]] \\ &= \mathbb{E}[\mathbb{E}[(x^n)^T(I - z_k z_k^T)^2 x^n|x_n]] \\ &= \mathbb{E}[(x^n)^T\left(\frac{1}{n}\sum_k (I - z_k z_k^T)^2\right)x^n] \end{aligned}$$

where $(I - z_k z_k^T)^2 = I - 2z_k z_k^T + z_k z_k^T z_k z_k^T = I - z_k z_k^T$ since $\|z_k\|^2 = 1$. So $\frac{1}{n}\sum_k (I - z_k z_k^T)^2 = \frac{1}{n}\sum_k (I - z_k z_k^T) = I - \frac{1}{n}\sum_k z_k z_k^T = I - Q = \frac{1}{2}I$. Therefore

$$\mathbb{E}\,\|x^{n+1}\|^2 = \frac{1}{2}\,\mathbb{E}\,\|x^n\|^2$$

and by induction

$$\mathbb{E}\,\|x^n\|^2 = \frac{1}{2^n}\|x^0\|^2 = \frac{1}{2^n}$$

(c) Consider the incremental gradient method with step size 1 initialized at the point $(0, 1)$. Compute

$$\|x_n - x_{opt}\|^2$$

when the increments are chosen in order, deterministically. That is, we first take a step with respect to the summand $k = 1$, then $k = 2$, all the way to $k = n$.

In this case, writing the consecutive updates yields

$$x^{n+1} = (I - z_n z_n^T)(I - z_{n-1} z_{n-1}^T) \dots (I - z_1 z_1^T) x^n$$

where each update $I - z_k z_k^T$ corresponds to a projection on the orthogonal of z_k. This can be verified by writing

$$\begin{aligned}
I - z_k z_k^T &= I - \begin{pmatrix} \cos^2 \frac{k\pi}{n} & \sin \frac{k\pi}{n} \cos \frac{k\pi}{n} \\ \sin \frac{k\pi}{n} \cos \frac{k\pi}{n} & \sin^2 \frac{k\pi}{n} \end{pmatrix} \\
&= \begin{pmatrix} \sin^2 \frac{k\pi}{n} & -\sin \frac{k\pi}{n} \cos \frac{k\pi}{n} \\ -\sin \frac{k\pi}{n} \cos \frac{k\pi}{n} & \cos^2 \frac{k\pi}{n} \end{pmatrix} \\
&= \begin{pmatrix} \cos^2 \frac{k\pi}{n} + \frac{\pi}{2} & \cos \frac{k\pi}{n} + \frac{\pi}{2} \sin \frac{k\pi}{n} + \frac{\pi}{2} \\ \cos \frac{k\pi}{n} + \frac{\pi}{2} \sin \frac{k\pi}{n} + \frac{\pi}{2} & \sin^2 \frac{k\pi}{n} + \frac{\pi}{2} \end{pmatrix} \\
&= \bar{z}_k \bar{z}_k^T
\end{aligned}$$

where $\bar{z}_k = \begin{pmatrix} \cos \frac{k\pi}{n} + \frac{\pi}{2} \\ \sin \frac{k\pi}{n} + \frac{\pi}{2} \end{pmatrix}$.

Now

$$x^{m+1} = \bar{z}_n \bar{z}_n^T \dots \bar{z}_1 \bar{z}_1^T x_m$$

where for all k, $\bar{z}_{k+1}^T \bar{z}_k$ is the dot product of two unit vectors that form a $\frac{\pi}{n}$ angle, i.e. $\bar{z}_{k+1}^T \bar{z}_k = \cos \frac{\pi}{n}$, and

$$x^{m+1} = (\cos \frac{\pi}{n})^{n-1} \bar{z}_n \bar{z}_1^T x_m$$

and $\bar{z}_n = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = x_0$. So $x^m = \|x^m\| x_0$, and

$$x^{m+1} = \|x^m\| (\cos \frac{\pi}{n})^{n-1} (\bar{z}_1^T x_0) x_0 = \|x^m\| (\cos \frac{\pi}{n})^n x_0$$

so $\|x^{m+1}\| = \|x^m\| (\cos \frac{\pi}{n})^n$ thus by induction

$$\|x^m\| = (\cos \frac{\pi}{n})^{mn}$$

2 In logistic regression, one is given a data set of pairs (x_i, y_i) where $x_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$. The goal is to find a vector w and scalar b such that

$$\sum_{i=1}^{n} -y_i(b + w^T x_i) + \log(1 + \exp(b + w^T x_i)) + \lambda \|w\|_2^2 \tag{1}$$

is minimized.

Program BFGS, LBFGS, and the Barzalai-Borwein methods. Test these methods using the function(1) on the adult data set with $\lambda = 10^{-3}$.

Use Weak Wolfe Line Search for the line search in BFGS and LBFGS. Use the stopping criterion

$$\|\nabla f(x)\|^2 \leq \texttt{toler} * (1 + |f(x)|)$$

Describe the optimal parameter settings for each algorithms to achieve the smallest number of function and gradient evaluations. Devise a methodology for getting the lowest possible computation time with some hybrid of these three methods. Submit your code as printout with comments.

BFGS

The BFGS iteration can be summarized as

- $p_k = -H_k \nabla f(x_k)$
- $x_{k+1} = x_k + \alpha_k p_k$ where α_k is chosen to satisfy the Wolfe condition: for some $0 < c_1 < c_2 < 1$:

$$f(x_{k+1}) \leq f(x_k) + c_1 \alpha_k \nabla f(x_k)^T p_k \qquad \text{(Armijo)}$$
$$\nabla f(x_{k+1})^T p_k \geq c_2 \nabla f(x_k)^T p_k \qquad \text{(Wolfe)}$$

- $s_k = x_{k+1} - x_k$
- $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$
- $H_{k+1} = (I - \rho_k s_k y_k^T)^T H_k (I - \rho_k s_k y_k^T) + \rho_k s_k s_k^T$ where $\rho_k = \frac{1}{y_k^T s_k}$

The Weak Wolfe line search can be implemented as: $t = 1$, $\alpha = 0$, $\beta = \infty$

- if Armijo condition fails, $\beta = t$, $t = \frac{1}{2}(\alpha + \beta)$
- else if Wolfe condition fails, $\alpha = t$, $t = 2\alpha$ if $\beta = \infty$ and $\frac{1}{2}(\alpha + \beta)$ otherwise.

The Matlab implementation is given below

```
function [fxp, xp]=BFGS(f, g, x0)

global toler;
global nbFcalls maxFcalls;

n = size(x0, 1);
I = eye(n);
nbFcalls = 0;

% initialize H and x
H = I;
x = x0;
% iterate
while (1)
    % compute the descent direction
    gx = g(x);
    p = -H*gx;

    % line search for alpha to satisfy the weak Wolfe conditions
    alpha = WW(f, g, x, p);

    % compute next iterate
    xp = x + alpha*p;

    % update the H matrix
    s = xp - x;
    gxp = g(xp);
    y = gxp - gx;
    rho = 1/(y'*s);
    Hu = (I - rho*s*y');
    H = Hu'*H*Hu + rho*(s*s');

    % compute the function value and update x
    fxp = f(xp)
    x = xp;

    % check stopping criterion
    e = norm(gxp)^2 - toler*(1+abs(fxp))
    if(e < 0 || (nbFcalls > maxFcalls))
        break
    end
end
```

Weak Wolfe line search

```
% f is the function handle, g the gradient handle, x the current point, d
% the search direction
function t = WW(f, g, x, p) %% WW for weak Wolfe

global c1 c2;
global fs;
t0 = 1;


beta = Inf;
alpha = 0;
t = t0;
found = 0;

% f(x)
ff = f(x);
% g(x)
gg = g(x);
while(¬found)
    % next point
    xp = x+t*p;
    ffp = f(xp);
    fs = [fs min(ff, ffp)]; % one gradient or function call per iteration of the line search
    %display({ff, ffp})
    if(ffp > ff + t*c1*gg'*p) % if the Armijo condition fails
        beta = t;
        t = (alpha + beta)/2;
    elseif (g(xp)'*p < c2*gg'*p) % if the Wolfe condition fails
        alpha = t;
        if(beta == Inf)
            t = 2*alpha;
        else
            t = (alpha+beta)/2;
        end
    else
        found = 1; % found a t which satisfies both conditions
    end
end
t % return t
```

LBFGS

The LBFGS iteration can be summarized as: instead of maintaining (and storing) the full matrix H_k, choose a horizon τ, then

- $q = \nabla f(x_k)$
- for $i \in \{k-1, \ldots, k-\tau\}$, $\alpha_i = \frac{s_i^T q}{s_i^T y_i}$, $q = q - \alpha_i y_i$
- $r = \lambda_k q$ where $\lambda_k = \frac{s_{k-1}^T s_{k-1}}{y_{k-1}^T s_{k-1}}$
- for $i \in \{k-\tau, \ldots, k-1\}$, $\beta_i = \frac{y_i^T r}{y_i^T s_i}$, $r = r + s_i(\alpha_i - \beta_i)$
- use r as descent direction.

The matlab implementation is given below

```
function [fxp, xp]=LBFGS(f, g, x0)

global toler;
global nbFcalls maxFcalls;
global tau;
n = size(x0, 1);

nbFcalls = 0;

% initialize the memory
%tau = 5; % remember the last 5 iterates
ss = zeros(n, tau);
ys = zeros(n, tau);
alphas = zeros(1, tau);
betas = zeros(1, tau);

% initialize x and lambda
x = x0;
lambda = 1;
k = 0;
% iterate
while (1)
    gx = g(x);
    % compute the descent direction
    q = gx;
    T = min(tau, k);
    for i=1:T;
        si = ss(:, i);
        yi = ys(:, i);
        alphas(i) = si'*q/(si'*yi);
        q = q - alphas(i)*yi;
    end
    r = lambda*q;
    for ii=1:T
        i = T-ii+1;
        yi = ys(:, i);
        si = ss(:, i);
        betas(i) = yi'*r/(yi'*si);
        r = r+si*(alphas(i) - betas(i));
    end

    % -r is the descent direction

    % line search for alpha to satisfy the weak Wolfe conditions
    alpha = WW(f, g, x, -r);

    % compute next iterate
    xp = x - alpha*r;

    % update the memory
    s = xp - x;
    gxp = g(xp);
    y = gxp - gx;

    lambda = s'*s / (y'*s);

    % shift all tables
    for i=0:tau-2
        ss(:, tau-i) = ss(:, tau-i-1);
        ys(:, tau-i) = ys(:, tau-i-1);
    end
    ss(:, 1) = s;
    ys(:, 1) = y;

```

```
65      % compute the function value and update x
66      fxp = f(xp)
67      x = xp;
68
69      % check stopping criterion
70      e = norm(gxp)^2 - toler*(1+abs(fxp))
71      if(e < 0 || (nbFcalls > maxFcalls))
72          break
73      end
74  end
```

BB

The Barzalai-Borwein method is simply

$$x_{k+1} = x_k - \alpha_k \nabla f(x_k)$$

where $\alpha_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}$. α_0 is a parameter.

The Matlab implementation is given below

```
1   function [fxp, xp]=BB(f, g, x0)
2
3   global nbFcalls maxFcalls;
4
5   global toler;
6   global fs;
7   global alpha0;
8   nbFcalls = 0;
9   fs = [fs, f(x0)];
10
11  % initialize alpha and x
12  alpha = alpha0;
13  x = x0;
14  % iterate
15  while (1)
16      % compute the descent direction
17      p = -g(x);
18
19      % compute next iterate
20      s = alpha*p;
21      xp = x + s;
22
23      % update alpha
24      gxp = g(xp);
25      y = gxp + p;
26      alpha = s'*y/(y'*y);
27
28      % compute the function value and update x
29      fxp = f(xp)
30      fs = [fs fxp];
31      x = xp;
32
33      % check stopping criterion
34      e = norm(gxp)^2 - toler*(1+abs(fxp))
35      if(e < 0 || (nbFcalls > maxFcalls))
36          break
37      end
38  end
```

Expression of the gradient

by augmenting all vectors X_i with $X_{i,0} = 1$, and writing $w_0 = b$, we have

$$f(x) = \sum_{i=1}^{n} -y_i w^T X_i + \log(1 + e^{w^T X_i}) + \lambda \|w\|^2$$

so

$$\begin{aligned} \frac{\partial f}{\partial w_j} &= \sum_{i=1}^{n} -y_i X_{i,j} + X_{i,j} \frac{e^{w^T X_i}}{1 + e^{w^T X_i}} + 2\lambda w_j \\ &= 2\lambda w_j + \sum_{i=1}^{n} X_{i,j} (\frac{e^{w^T X_i}}{1 + e^{w^T X_i}} - y_i) \end{aligned}$$

so

$$\nabla_w f = 2\lambda w + X^T z$$

where $z_i = \frac{e^{w^T X_i}}{1+e^{w^T X_i}} - y_i$

The Matlab implementation of the loss function and the gradient function is given below

```
function l = loss(w)

lambda = .001;
global X;
global y;
global nbFcalls;

nbFcalls = nbFcalls+1;
wo = w;
wo(1) = 0;
l = lambda*norm(wo)^2;
n = size(X, 2);
for i=1:n
    wXi = w'*X(:, i);
    l = l - y(i)*wXi + log(1+exp(wXi));
end
```

```
function g = grad(w)

lambda = .001;
global X y nbGcalls;

nbGcalls = nbGcalls+1;

wo = w;
wo(1) = 0;
n = size(X, 2);
z = zeros(n, 1);
for i=1:n
    Xi = X(:, i);
    wXi = w'*Xi;
    z(i) = exp(wXi)/(1+exp(wXi)) - y(i);
end

g = 2*lambda*wo + X*z;
```

Results

To find the optimal parameter setting for each method, we run, for different values of the parameters, the algorithm for a fixed number of steps (a step being one function or one gradient evaluation).

The code is given at the end.

Optimal parameters:

- BFGS: $c_1 = .4$, $c_2 = .5$
- LBFGS: $c_1 = .4$, $c_2 = .5$, memory: $\tau = 4$.
- Barzalai-Borwein: $\alpha_0 = 10^{-4}$

The figure below shows the losses as a function of step number for the optimal parameter choice for each method. For `toler` $= 10^{-3}$, we have the following convergence (number of function evaluations)

- BFGS: after 506 iterations, $f(x_{\text{best}}) = 1.0515e + 04$
- LBFGS: after 514 iterations, $f(x_{\text{best}}) = 1.0512e + 04$
- BB: 269 iterations, $f(x_{\text{best}}) = 1.0510e + 04$

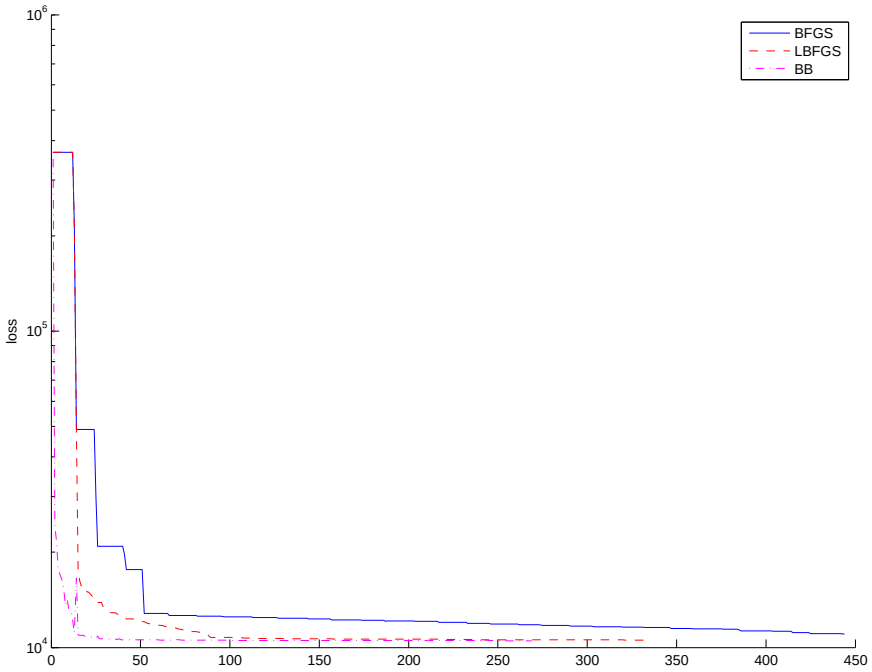

Figure 1: Loss as a function of iteration number for the optimal parameter choice. BB is not a descent method (no line search).

Combining the algorithms

It seems that the BB method is fast to converge within a ball of the optimum, and then yields slower convergence. We can use it first, and when we get closer to the optimum, switch to either BFGS or LBFGS to have a better approximation of the function around the equilibrium.

An algorithm combining both is given below, where the switch happens at $\|\nabla f(x)\|^2 \leq$ `toler`$*(1+|f(x)|)$, then we decrease `toler` to 10^{-4} and run LBFGS.

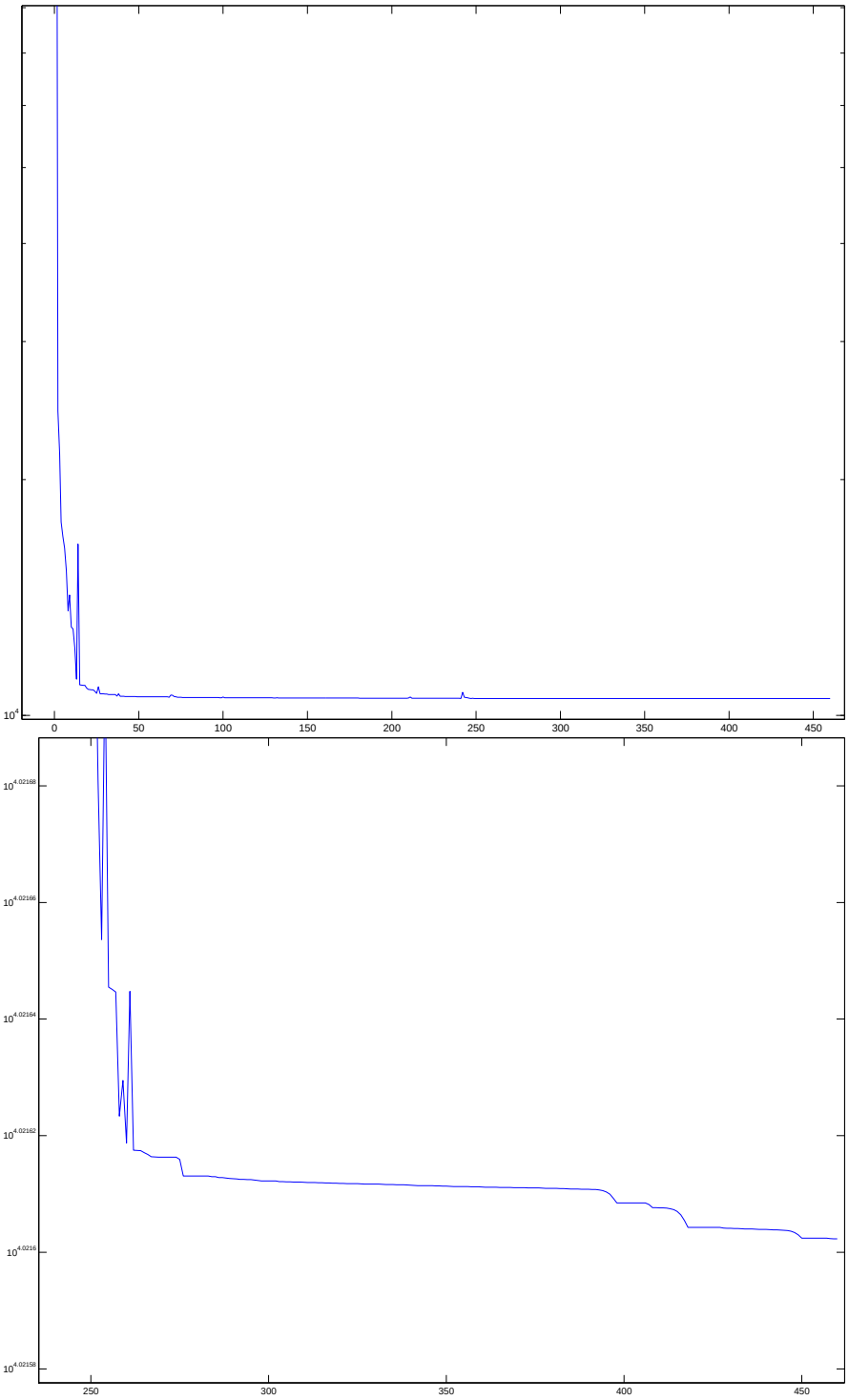

Figure 2: Loss of the combined method, and zoomed in plot around the point where the switch happens. After the switch, the decrease becomes slower, but we do have a marginal improvement by using LBFGS after BB.

```
n = size(X0, 2);
p = size(X0, 1);

% training set
global X y;
X = [ones(1, n); X0]; % augment the training samples with a 1 on their first entry
y = y;

% parameters of the algorithms
global c1 c2 toler tau alpha0;
toler = .001;
% Weak Wolfe line search parameters: c1, c2
% LBFGS parameter: tau
% BB paramater: alpha0
% number of function calls and gradient calls
global nbFcalls nbGcalls maxFcalls;
nbFcalls = 0;
nbGcalls = 0;

% sequence of functions values f(x_k)
global fs;

% initial point
w0 = ones(p+1, 1);

%%-------------------------------------------------------------------------
%% compare performance of BFGS with different parameters
maxFcalls = 200;
fBest = Inf;
fcs = [];

c1s = [.1, .2, .3, .4, .5];
c2s = [.4, .5, .6, .7, .8];
for c1 = c1s
    for c2 = c2s
        if(c1 < c2)
            [fx, x] = BFGS(@loss, @grad, w0);
            fcs = [fcs fx];
            if (fx < fBest)
                fBest = fx;
                cBest = [c1, c2];
            end
        end
    end
end

%%-------------------------------------------------------------------------
%% compare performance of LBFGS with different parameters
maxFcalls = 400;
fBest = Inf;
fcs = [];

c1s = [.2, .4];
c2s = [.6, .8];
taus = [4, 8];
for c1 = c1s
    for c2 = c2s
        for tau=taus
            [fx, x] = LBFGS(@loss, @grad, w0);
            fcs = [fcs fx];
            if (fx < fBest)
                fBest = fx;
                cBest = [c1, c2, tau];
            end
```

```
        end
    end
end

%%-------------------------------------------------------------------------
%% compare performance of BB with different parameters
maxFcalls = 400;
fBest = Inf;
w0 = ones(p+1, 1);

alpha0s = [.0001, .000001];
for alpha0 = alpha0s
    fx = BB(@loss, @grad, w0);
    if (fx < fBest)
        fBest = fx;
        cBest = [alpha0];
    end
end

%%-------------------------------------------------------------------------
%% Compare performance of the different methods with optimal parameters
maxFcalls = Inf;
toler = .001;
w0 = ones(p+1, 1);

figure(1)
hold on

% BFGS
fs = [];
c1 = .4;
c2 = .5;
[fx, x] = BFGS(@loss, @grad, w0)
nbFcalls
plot(fs, 'b')

% LBFGS
fs = [];
c1 = .4;
c2 = .5;
tau = 4;
[fx, x] = LBFGS(@loss, @grad, w0)
nbFcalls
plot(fs, 'r--')

% BB
fs = [];
alpha0 = .0001;
[fx, x] = BB(@loss, @grad, w0)
nbFcalls
plot(fs, 'm-.')

ylabel('loss')
legend({'BFGS', 'LBFGS', 'BB'})

%%-------------------------------------------------------------------------
%% Combined
maxFcalls = Inf;
toler = .001;
w0 = ones(p+1, 1);

figure(1)
hold on

% BB
```

```
fs = [];
[fx, x] = BB(@loss, @grad, w0)

% then LBFGS
toler = .0001;
[fx, x] = LBFGS(@loss, @grad, x)

plot(fs, 'b')
ylabel('loss')

%%-------------------------------------------------------------------------
set(gca,'YScale','log');
savefig('losses_combined_zoom', 'pdf')
```