THE UNIVERSITY OF TEXAS
AT AUSTIN

EE381V LARGE SCALE OPTIMIZATION

Problem Set 2

Edited by LaTeX

Department of Computer Science

STUDENT
Jimmy Lin
xl5224

COURSE COORDINATOR
Sujay Sanghavi

UNIQUE NUMBER
17350

RELEASE DATE
September 12, 2014

DUE DATE
September 18, 2014

TIME SPENT
10 hours

September 13, 2014

Table of Contents

List of Figures

1 Matlab and Computational Assignment

1.1 Five flavors for Eq. (9.20) in B & V

1.1.1 Standard Gradient Descent with Backtracking

Command to be executed in matlab:

```
>> x_init = [1 1]'; alpha = 0.3; bta = 0.8;
>> [x, iter, all_costs] = gd_btls(x_init, @func, @func_grad, alpha, bta);
```

Dump

```
Iter: 1, Cost: 5.292007e+00, Conv_Rate: 0.106142, gamma: 0.800000
Iter: 2, Cost: 3.936296e+00, Conv_Rate: 0.743819, gamma: 0.409600
Iter: 3, Cost: 3.440868e+00, Conv_Rate: 0.874138, gamma: 0.327680
Iter: 4, Cost: 3.047567e+00, Conv_Rate: 0.885697, gamma: 0.262144
Iter: 5, Cost: 2.851476e+00, Conv_Rate: 0.935657, gamma: 0.262144
Iter: 6, Cost: 2.698084e+00, Conv_Rate: 0.946206, gamma: 0.209715
Iter: 7, Cost: 2.621951e+00, Conv_Rate: 0.971783, gamma: 0.134218
Iter: 8, Cost: 2.589490e+00, Conv_Rate: 0.987619, gamma: 0.107374
Iter: 9, Cost: 2.571558e+00, Conv_Rate: 0.993075, gamma: 0.068719
                                  ...
                                  ...
Iter: 40, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.000000
Iter: 41, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.000000
Iter: 42, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.000000
Iter: 43, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.000000
Iter: 44, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.000000
Convergence reached!
```

Minima

```
x = [-0.3379, -0.0031], obj = 2.559267
```

Plot

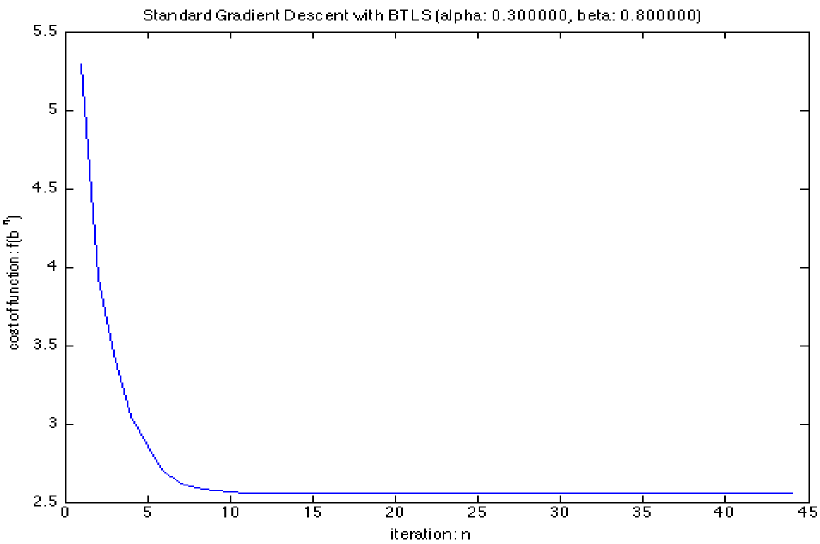

Figure 1: Standard gradient descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.1.2 Steepest Descent with P_1

Command to be executed in matlab:

```
>> P1 = [8 0; 0 2];
>> x_init = [1 1]'; alpha = 0.3; bta = 0.8;
>> [x, iter, all_costs] = sd_btls(x_init, @func, @func_grad, P1, alpha, bta);
```

Dump

```
Iter: 1, Cost: 4.109800e+00, Conv_Rate: 0.082430, gamma: 0.035184
Iter: 2, Cost: 2.574134e+00, Conv_Rate: 0.626340, gamma: 0.409600
Iter: 3, Cost: 2.561394e+00, Conv_Rate: 0.995051, gamma: 1.000000
Iter: 4, Cost: 2.559319e+00, Conv_Rate: 0.999190, gamma: 0.800000
Iter: 5, Cost: 2.559268e+00, Conv_Rate: 0.999980, gamma: 0.800000
Iter: 6, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 7, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 8, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 9, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 10, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 11, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.800000
Iter: 12, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 1.000000
Convergence reached!
```

Minima

```
x = [-0.3466 -0.0000], obj = 2.559267
```

Plot

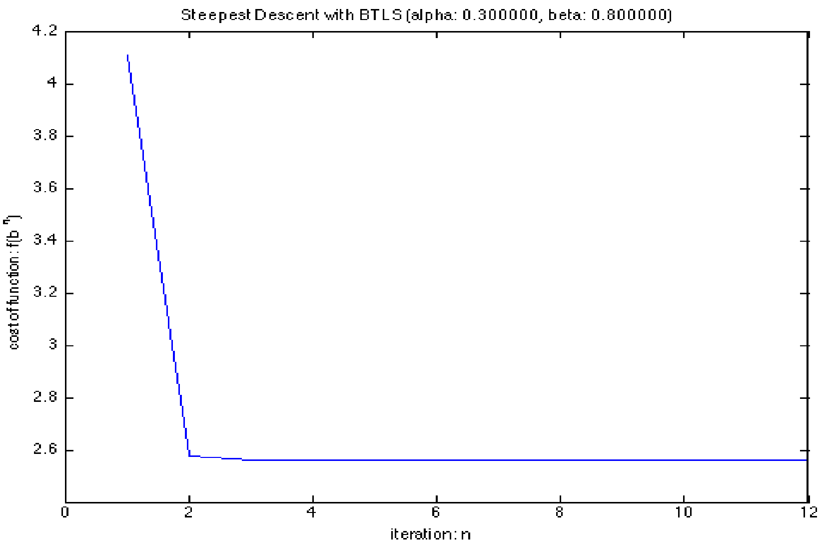

Figure 2: Steepest Descent with BTLS on Eq. 9.20 with P_1, $\alpha = 0.3$ and $\beta = 0.8$

1.1.3 Steepest Descent with P_2

Command to be executed in matlab:

```
>> P2 = [2 0; 0 8];
>> x_init = [1 1]'; alpha = 0.3; bta = 0.8;
>> [x, iter, all_costs] = sd_btls(x_init, @func, @func_grad, P2, alpha, bta);
```

Dump

```
Iter: 1, Cost: 5.397521e+00, Conv_Rate: 0.108258, gamma: 0.011529
Iter: 2, Cost: 4.867283e+00, Conv_Rate: 0.901763, gamma: 0.068719
Iter: 3, Cost: 4.673428e+00, Conv_Rate: 0.960172, gamma: 0.107374
Iter: 4, Cost: 4.447617e+00, Conv_Rate: 0.951682, gamma: 0.085899
Iter: 5, Cost: 4.276423e+00, Conv_Rate: 0.961509, gamma: 0.134218
Iter: 6, Cost: 4.100684e+00, Conv_Rate: 0.958905, gamma: 0.107374
Iter: 7, Cost: 3.955941e+00, Conv_Rate: 0.964703, gamma: 0.134218
Iter: 8, Cost: 3.820349e+00, Conv_Rate: 0.965724, gamma: 0.134218
Iter: 9, Cost: 3.687908e+00, Conv_Rate: 0.965333, gamma: 0.134218
Iter: 10, Cost: 3.560463e+00, Conv_Rate: 0.965442, gamma: 0.134218
Iter: 11, Cost: 3.444372e+00, Conv_Rate: 0.967394, gamma: 0.134218
Iter: 12, Cost: 3.347890e+00, Conv_Rate: 0.971989, gamma: 0.167772
Iter: 13, Cost: 3.259404e+00, Conv_Rate: 0.973570, gamma: 0.167772
                                ...
                                ...
Iter: 164, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.262144
Iter: 165, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.409600
Iter: 166, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.327680
Iter: 167, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.262144
Iter: 168, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.167772
Convergence reached!
```

Minima

```
x = [-0.3466 -0.0000], obj = 2.559267
```

Plot

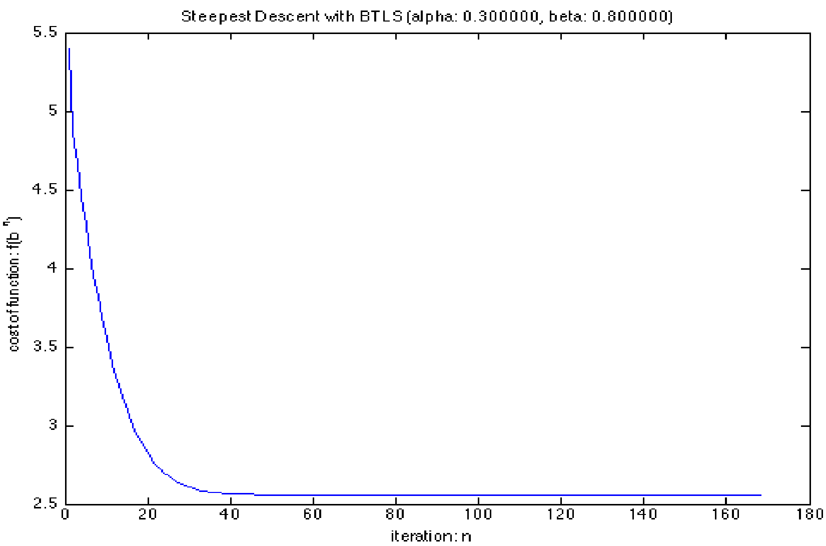

Figure 3: Steepest Descent with BTLS on Eq. 9.20 with P_2, $\alpha = 0.3$ and $\beta = 0.8$

1.1.4 Cyclic Coordinate Descent

Command to be executed in matlab:

```
>> x_init = [1 1]'; alpha = 0.3; bta = 0.8;
>> [x, iter, all_costs] = ccd_btls(x_init, @func, @func_grad, alpha, bta);
```

Dump

```
Iter: 1, Cost: 1.217687e+01, Conv_Rate: 0.244232, gamma: 0.043980
Iter: 2, Cost: 3.551842e+00, Conv_Rate: 0.071239, gamma: 0.035184
Iter: 3, Cost: 3.064142e+00, Conv_Rate: 0.862691, gamma: 0.209715
Iter: 4, Cost: 2.732279e+00, Conv_Rate: 0.769257, gamma: 0.134218
Iter: 5, Cost: 2.664867e+00, Conv_Rate: 0.975328, gamma: 0.209715
Iter: 6, Cost: 2.600923e+00, Conv_Rate: 0.951925, gamma: 0.134218
Iter: 7, Cost: 2.585633e+00, Conv_Rate: 0.994121, gamma: 0.262144
Iter: 8, Cost: 2.563802e+00, Conv_Rate: 0.985727, gamma: 0.107374
Iter: 9, Cost: 2.561867e+00, Conv_Rate: 0.999245, gamma: 0.209715
Iter: 10, Cost: 2.560171e+00, Conv_Rate: 0.998584, gamma: 0.107374
Iter: 11, Cost: 2.559836e+00, Conv_Rate: 0.999869, gamma: 0.167772
Iter: 12, Cost: 2.559551e+00, Conv_Rate: 0.999758, gamma: 0.107374
Iter: 13, Cost: 2.559478e+00, Conv_Rate: 0.999971, gamma: 0.167772
                                 ...
                                 ...
Iter: 60, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.134218
Iter: 61, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.262144
Iter: 62, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.085899
Iter: 63, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.327680
Iter: 64, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.134218
Convergence reached!
```

Minima

```
x = [-0.3466 0.0000], obj = 2.559267
```

Plot

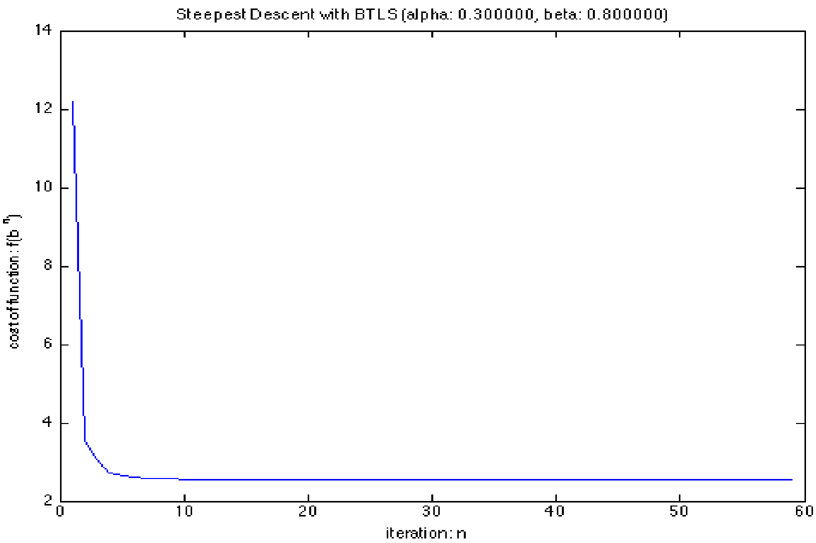

Figure 4: Cyclical Coordinate Descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.1.5 Greedy Coordinate Descent

Command to be executed in matlab:

```
>> x_init = [1 1]'; alpha = 0.3; bta = 0.8;
>> [x, iter, all_costs] = gcd_btls(x_init, @func, @func_grad, alpha, bta);
```

Dump

```
Iter: 1, Cost: 5.615225e+00, Conv_Rate: 0.112625, gamma: 0.005903
Iter: 2, Cost: 4.227075e+00, Conv_Rate: 0.752788, gamma: 0.028147
Iter: 3, Cost: 3.306101e+00, Conv_Rate: 0.782125, gamma: 0.035184
Iter: 4, Cost: 2.823271e+00, Conv_Rate: 0.853958, gamma: 0.054976
Iter: 5, Cost: 2.666299e+00, Conv_Rate: 0.944401, gamma: 0.068719
Iter: 6, Cost: 2.577867e+00, Conv_Rate: 0.966834, gamma: 0.107374
Iter: 7, Cost: 2.566546e+00, Conv_Rate: 0.995608, gamma: 0.107374
Iter: 8, Cost: 2.561822e+00, Conv_Rate: 0.998160, gamma: 0.107374
Iter: 9, Cost: 2.560570e+00, Conv_Rate: 0.999511, gamma: 0.107374
Iter: 10, Cost: 2.559369e+00, Conv_Rate: 0.999531, gamma: 0.107374
Iter: 11, Cost: 2.559298e+00, Conv_Rate: 0.999972, gamma: 0.107374
Iter: 12, Cost: 2.559277e+00, Conv_Rate: 0.999992, gamma: 0.107374
                                      ...
                                      ...
Iter: 32, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.107374
Iter: 33, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.107374
Iter: 34, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.107374
Iter: 35, Cost: 2.559267e+00, Conv_Rate: 1.000000, gamma: 0.107374
Convergence reached!
```

Minima

```
x = [-0.3466 -0.0000], obj = 2.559267
```

Plot

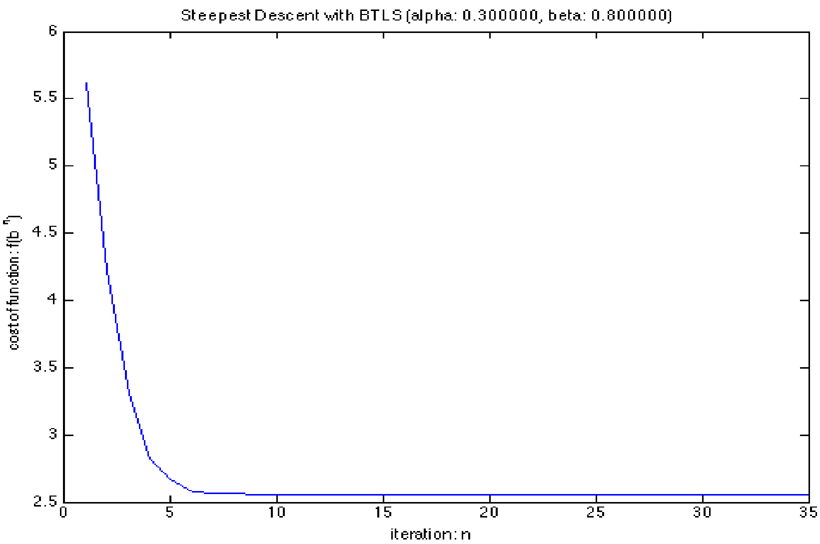

Figure 5: Greedy Coordinate Descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.1.6 Conclusions

- $(1,1)$ is an decent initial point for all five flavors of optimization methods.
- Steepest descent method could both enhance and impair the convergence speed, comparing to standard gradient descent (uniform heuristic or unheuristic). The specific effect depends on what heuristic matrix is provided.
- Greedy coordinate descent does converge to optima in less number of iterations than the cyclic cooridnate descent but in larger computational cost in each iteration.

2 Written Problems

A Codes Printout

A.1 Eq. 20 and its gradient

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% objective function for optimization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    y = func (x)
% Parameter:
%    x: input vector, must be column vector

function y = func (x)
assert (size(x, 1) == 2);
assert (size(x, 2) == 1);
x_1 = x(1);
x_2 = x(2);
term1 =    x_1 +3*x_2 -0.1;
term2 =    x_1 -3*x_2 -0.1;
term3 = -1*x_1        -0.1;
y = exp(term1) + exp(term2) + exp(term3);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% gradient of function EQ. 9.20 in B & V
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    gradient = func_grad (x)
% Parameter:
%    b: variable vector

function gradient = func_grad (x)
assert(all(size(x)==[2 1]))
x_1 = x(1);
x_2 = x(2);
grad_1 = exp(x_1+3*x_2-0.1) + exp(x_1-3*x_2-0.1) -1*exp(-1*x_1-0.1);
grad_2 = 3*exp(x_1+3*x_2-0.1) -3*exp(x_1-3*x_2-0.1);
gradient = [grad_1 grad_2]';
end
```

A.2 Standard Gradient Descent with BackTracking Line Search

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% HW2: Gradient Descent with Backtrack Line Search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    [b, iter, all_costs] = gd_btls (b_init, f, fgrad, alpha, discount)
% Parameters:
%    b_init: inital value of variable
%    f: objective function
%    fgrad: gradient of objective function
%    alpha: parameter for evaluating "OK" step size
%    discount(beta): discounting factor for not-OK step size
%% Note that the distinguisable period eps = 10e-16

function [b, iter, all_costs] = gd_btls (b_init, f, fgrad, alpha, discount)
assert (discount > 0 && discount < 1);
assert (alpha > 0 && alpha < 0.5);
iter = 1; % iteration count
b = b_init;  % variable vector
last_cost = func(b);  % value of objective function in most recent iteration
all_costs = []; % all values of objective function, for plotting
while true,
    %% compute essential numerics and do gradient descent
    gradient = fgrad(b);
    delta_b = -1.0 * gradient / norm(gradient);
    %% do backtrack line search
    gamma = 1.0;  % step size
    while f(b+gamma*delta_b) > f(b) + alpha*gamma*gradient'*delta_b,
        gamma = discount * gamma;
        % disp(sprintf('BTLS: new gamma is %f', gamma));
    end
    b = b + gamma * delta_b;
    cost = func(b);
    rate = (cost / last_cost);
    all_costs = [all_costs cost];
    %% output numeric information of this iteration
    disp (sprintf('Iter: %d, Cost: %e, Conv_Rate: %f, gamma: %f',iter,cost,rate,gamma));
    %% quadratic optimization converges to zero
    if abs((cost - last_cost) / last_cost) < eps,
        disp('Convergence reached!')
        break
    end
    %% prepare for next iteration
    last_cost = cost;
    iter = iter + 1;
end
%% uncomment following code for plotting individual gradient descent run
% plot f(b^(n)) with regard to n
plot (1:iter, all_costs)
title (sprintf ('Standard Gradient Descent with BTLS (alpha: %f, beta: %f)',alpha,discount));
xlabel ('iteration: n');
ylabel ('cost of function: f(b^n)');
end
```

A.3 Steepest Descent with BackTracking Line Search

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Steepest Descent Algorithm with Backtrack Line Search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    [b, iter, all_costs] = sd_btls (b_init, f, fgrad, P, alpha, discount)
% Parameters:
%    b_init: starting search point of variable
%    f: objective function
%    fgrad: gradient of objective function
%    P: matrix that defines norm of steepest descent
%    alpha: parameter for evaluating "OK" step size
%    discount(beta): discounting factor for not-OK step size
%% Note that the distinguisable period eps = 10e-16

function [b, iter, all_costs] = sd_btls (b_init, f, fgrad, P, alpha, discount)
assert (discount > 0 && discount < 1);
assert (alpha > 0 && alpha < 0.5);
iter = 1; % iteration count
b = b_init;  % variable vector
last_cost = func(b);  % value of objective function in most recent iteration
all_costs = []; % all values of objective function, for plotting
while true,
    %% compute essential numerics and do gradient descent
    gradient = fgrad(b);
    delta_b = -1.0 * inv(P) * gradient;
    %% do backtrack line search
    gamma = 1.0;
    while f(b+gamma*delta_b) > f(b) + alpha*gamma*gradient'*delta_b,
        gamma = discount * gamma;
        % disp(sprintf('BTLS: new gamma is %f', gamma));
    end
    b = b + gamma * delta_b;
    cost = func(b);
    rate = (cost / last_cost);
    all_costs = [all_costs cost];
    %% output numeric information of this iteration
    disp (sprintf('Iter: %d, Cost: %e, Conv_Rate: %f, gamma: %f',iter,cost,rate,gamma));
    %% quadratic optimization converges to zero
    if abs((cost - last_cost) / last_cost) < eps,
        disp('Convergence reached!')
        break
    end
    %% prepare for next iteration
    last_cost = cost;
    iter = iter + 1;
end
%% uncomment following code for plotting individual gradient descent run
% plot f(b^(n)) with regard to n
plot (1:iter, all_costs)
title (sprintf ('Steepest Descent with BTLS (alpha: %f, beta: %f)',alpha,discount));
xlabel ('iteration: n');
ylabel ('cost of function: f(b^n)');
end
```

A.4 Cyclic Coordinate Descent

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Cyclic Coordinate Descent Algorithm with Backtrack Line Search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    [b, iter, all_costs] = ccd_btls (b_init, f, fgrad, alpha, discount)
% Parameters:
%    b_init: starting search point of variable
%    f: objective function
%    fgrad: gradient of objective function
%    P: matrix that defines norm of steepest descent
%    alpha: parameter for evaluating "OK" step size
%    discount(beta): discounting factor for not-OK step size
% Note:
%    a) the minimal distinguisable value eps = 10e-16
%    b) coordinate descent in cyclical epoch is one iteration

function [b, iter, all_costs] = ccd_btls (b_init, f, fgrad, alpha, discount)
assert (discount > 0 && discount < 1);
assert (alpha > 0 && alpha < 0.5);
iter = 1; % iteration count
b = b_init;  % variable vector
ndim = size(b, 1);
last_cost = func(b);  % value of objective function in most recent iteration
all_costs = []; % all values of objective function, for plotting
while true,
    gradient = fgrad(b);
    for d = 1:ndim,
        %% compute essential numerics and do gradient descent
        delta_b = zeros (d, 1);
        delta_b(d) = -1.0 * gradient(d);
        %% do backtrack line search
        gamma = 1.0;
        while f(b+gamma*delta_b) > f(b) + alpha*gamma*gradient'*delta_b,
            gamma = discount * gamma;
        end
        b = b + gamma * delta_b;
    end
    cost = func(b);
    rate = (cost / last_cost);
    all_costs = [all_costs cost];
    %% output numeric information of this iteration
    disp (sprintf('Iter: %d, Cost: %e, Conv_Rate: %f, gamma: %f',iter,cost,rate,gamma));
    %% quadratic optimization converges to zero
    if abs((cost - last_cost) / last_cost) < eps,
        disp('Convergence reached!')
        break
    end
    %% prepare for next iteration
    iter = iter + 1;
    last_cost = cost;
end
%% uncomment following code for plotting individual gradient descent run
% plot f(b^(n)) with regard to n
plot (1:iter, all_costs)
title (sprintf ('Cyclic Coordinate Descent with BTLS (alpha: %f, beta: %f)',alpha,discount));
xlabel ('iteration: n');
ylabel ('cost of function: f(b^n)');
end
```

A.5 Greedy Coordinate Descent

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Greedy Coordinate Descent Algorithm with Backtrack Line Search
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Usage:
%    [b, iter, all_costs] = gcd_btls (b_init, f, fgrad, alpha, discount)
% Parameters:
%    b_init: starting search point of variable
%    f: objective function
%    fgrad: gradient of objective function
%    P: matrix that defines norm of steepest descent
%    alpha: parameter for evaluating "OK" step size
%    discount(beta): discounting factor for not-OK step size
% Note:
%    a) the minimal distinguisable value eps = 10e-16

function [b, iter, all_costs] = gcd_btls (b_init, f, fgrad, alpha, discount)
assert (discount > 0 && discount < 1);
assert (alpha > 0 && alpha < 0.5);
iter = 1; % iteration count
b = b_init;  % variable vector
ndim = size(b, 1);
last_cost = func(b);  % value of objective function in most recent iteration
all_costs = []; % all values of objective function, for plotting
while true,
    gradient = fgrad(b);
    waitlist = zeros(ndim, 1);
    gammalist = zeros(ndim, 1);
    for d = 1:ndim,
        %% compute essential numerics and do gradient descent
        delta_b = zeros (d, 1);
        delta_b(d) = -1.0 * gradient(d);
        %% do backtrack line search
        gamma = 1.0;
        while f(b+gamma*delta_b) > f(b) + alpha*gamma*gradient'*delta_b,
            gamma = discount * gamma;
        end
        waitlist(d) = f(b+gamma*delta_b);
        gammalist(d) = gamma;
    end
    [min_value, min_index] = min(waitlist);
    delta_b = zeros (d, 1);
    delta_b(min_index) = -1.0 * gradient(min_index);
    b = b + gammalist(min_index) * delta_b;
    cost = func(b);
    rate = (cost / last_cost);
    all_costs = [all_costs cost];
    %% output numeric information of this iteration
    disp (sprintf('Iter: %d, Cost: %e, Conv_Rate: %f, gamma: %f',iter,cost,rate,gamma));
    %% quadratic optimization converges to zero
    if abs((cost - last_cost) / last_cost) < eps,
        disp('Convergence reached!')
        break
    end
    %% prepare for next iteration
    iter = iter + 1;
    last_cost = cost;
end
%% uncomment following code for plotting individual gradient descent run
% plot f(b^(n)) with regard to n
plot (1:iter, all_costs)
title (sprintf ('Greedy Steepest Descent with BTLS (alpha: %f, beta: %f)',alpha,discount));
xlabel ('iteration: n');
ylabel ('cost of function: f(b^n)');
end
```