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

October 7, 2014

Table of Contents

List of Figures

Part I
Matlab and Computational Assignment

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

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

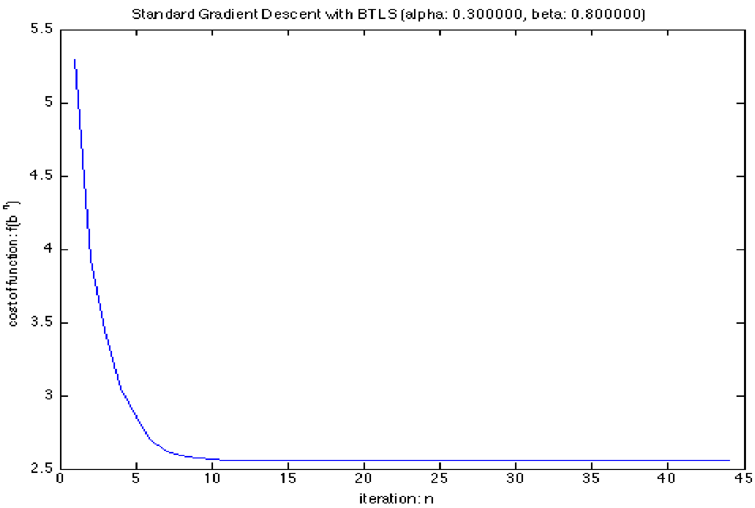

Figure 1: Standard gradient descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.2 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

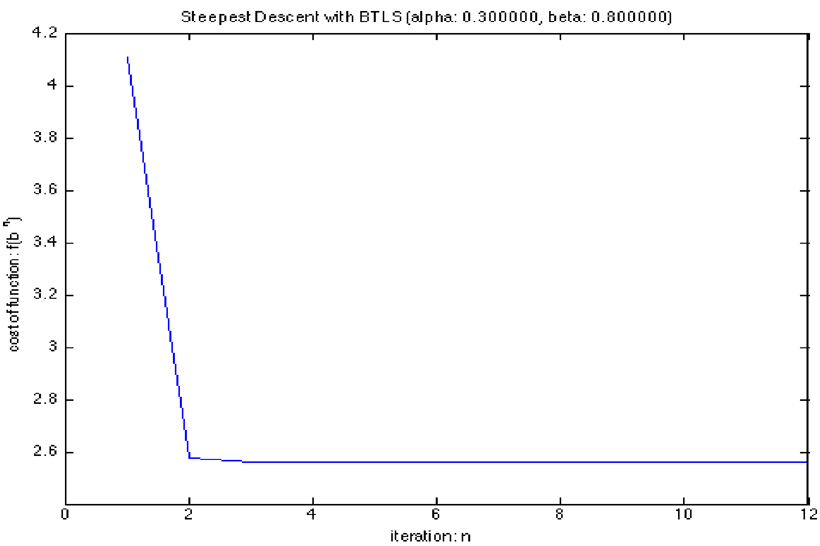

Figure 2: Steepest Descent with BTLS on Eq. 9.20 with P_1, $\alpha = 0.3$ and $\beta = 0.8$

1.3 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

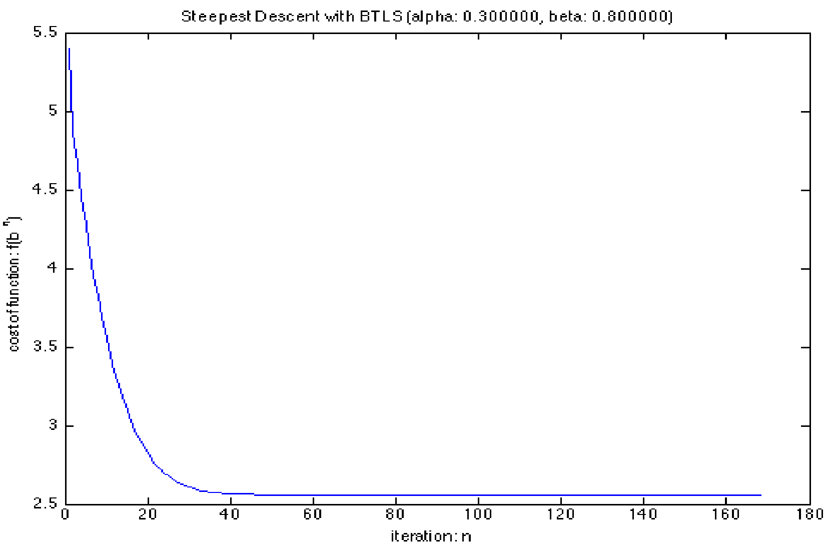

Figure 3: Steepest Descent with BTLS on Eq. 9.20 with P_2, $\alpha = 0.3$ and $\beta = 0.8$

1.4 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

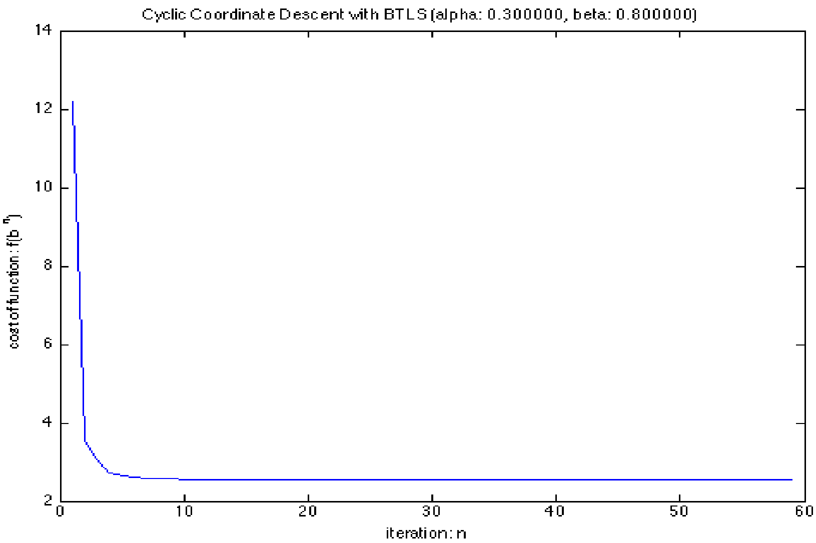

Figure 4: Cyclical Coordinate Descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.5 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

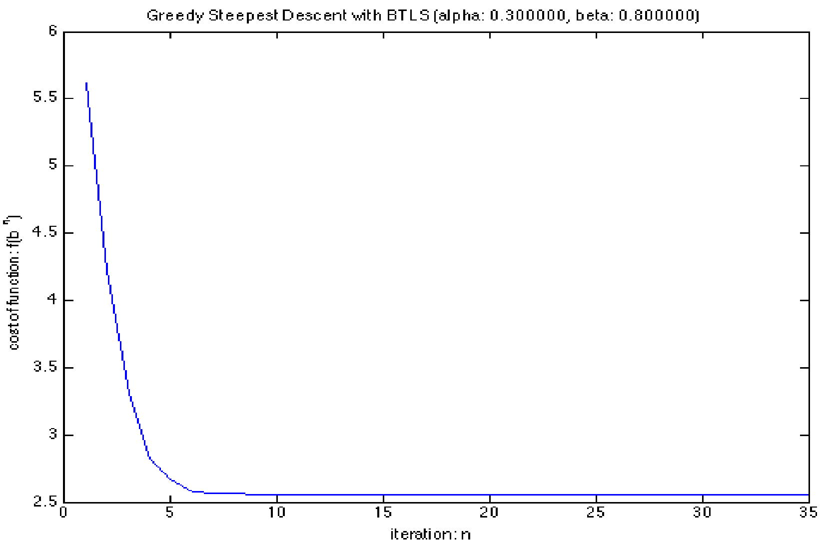

Figure 5: Greedy Coordinate Descent with BTLS on Eq. 9.20 with $\alpha = 0.3$ and $\beta = 0.8$

1.6 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.

Part II
Written Problems

1 Coordinate Descent

(a) Give an example

The example to illustrate failure of coordinate descent in converging to global minimum of function f at point x is

$$f(x, y) = ||(x, y)||_\infty = max(x, y) \tag{1}$$

at point $(x, y) = (-2, -2)$.
And the contour is drawn as follows:

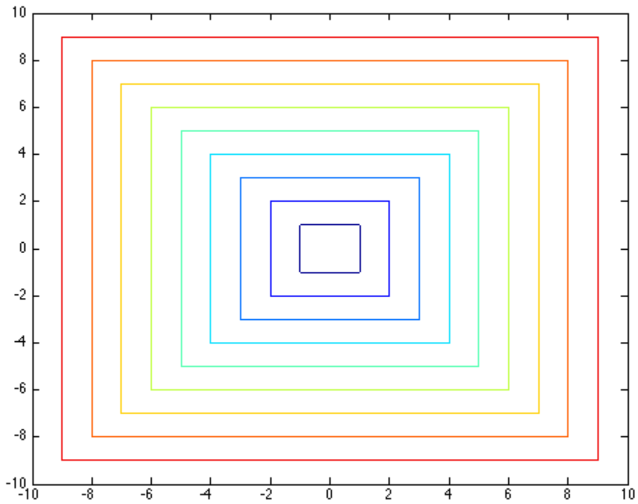

Figure 6: Contour of function $f(x, y) = ||(x, y)||_\infty$

Note that since $f(x, y) = ||(x, y)||_\infty$ is not differentiable, any coordinate descent that employs gradient of $f(x, y)$ is not available. But we can still apply non-gradient-based coordinate descent algorithm on this problem.

Remark 1. *There is a bug for this question. Function can be chosen as $f(x) = x$, which is convex but not strongly convex and the point x can be arbitrary point. Since $f(x) = x$ has its global minimum $-\infty$, then the coordinate descent starting from arbitrary point $x \in \mathbb{R}$ can never get to the global minimum of f forever.*

(b) $f(x, y) = x^2 + y^2 + 3xy$

The coordinate descent with exact line search **will not** always converge to a stationary point. The stationary point is $(x, y) = (0, 0)$. Consider the coordinate descent at $(10, -10)$:

$$\frac{\partial f(x, y)}{\partial x} = 2x + 3y = -10 \qquad f(x, y) \text{ will descend if } x \text{ increase} \tag{2}$$

$$\frac{\partial f(x, y)}{\partial y} = 2y + 3x = 10 \qquad f(x, y) \text{ will descend if } y \text{ decrease} \tag{3}$$

Obviously, the coordinate descent at $(10, -10)$ will go away from $(0, 0)$. Therefore, $f(x, y)$ has no guarantee to converge to stationary point by coordinate descent with exact line search.

2 Condition Number

For arbitrary step size t, the strongly convex function $f(x) = \frac{1}{t}x^2$ starting from $x_0 = 1$ does not converge to optimal solution.

Proof. We begin with showing $f(x) = \frac{1}{t}x^2$ is strongly convex. Obviously, $0 \leq f(x) \leq \frac{2}{t}$. Hence, function $f(x) = \frac{1}{t}x^2$ is strongly convex with $m = 0$ and $M = \frac{1}{t}$. (The hessian of $f(x)$ is bounded.)

Then we show that $f(x) = \frac{1}{t}x^2$ starting from $x_0 = 1$ does not converge to optimal solution. The gradient of $f(x)$ at x_0 is given by

$$\nabla f(x_0) = \nabla f(x)|_{x=1} = \frac{2}{t}x|_{x=1} = \frac{2}{t} \tag{4}$$

By gradient algorithm, the update direction at x_0 is given by

$$\delta x_0 = -\nabla f(x_0) = \frac{2}{t} \tag{5}$$

Apply the update step

$$x^+ = x_0 - \Delta x_0 = 1 - t \cdot \frac{2}{t} = 1 - 2 = -1 \tag{6}$$

It is obvious that

$$f(x^+) = f(-1) = \frac{1}{t} = f(1) = f(x_0) \tag{7}$$

Continue the gradient algorithm repeatedly. And then we find that the function at k-step always equals its initial value $f(x_0)$ and never reach its optimum.

$$f(x_k) = f(x_0) \neq f(0) = 0 \tag{8}$$

Hence, it is proved that *for any fixed step size t , there exists a smooth strongly convex function with bounded Hessian, such that a fixed stepsize gradient algorithm starting from some point x_0, does not converge to the optimal solution.* □

3 Decreasing Stepsize

Proof. Let constant m and M be the lower and uppper bound on hessian of $f(x)$. That is

$$mI \leq \nabla^2 f(x) \leq MI \tag{9}$$

The given conditions are

$$\lim_{k\to\infty} t_k = 0 \tag{10}$$

$$\Sigma_{k=0}^{\infty} t_k = \infty \tag{11}$$

which tells us that t_k will start from very large value and finally decrease to 0 (sufficiently small step size). Then we have

$$\exists k,\ 0 < t_k < \frac{1}{M} \tag{12}$$

By second-order approximation and lower bound of Hessian on step k, we have

$$f(x_{k+1}) \leq f(x) - t_k||f(x_k)||_2^2 + \frac{Mt_k^2}{2}||f(x_k)||_2^2 \tag{13}$$

$$f(x_{k+1}) \leq f(x)(\frac{Mt_k}{2} - 1)t_k||f(x_k)||_2^2 \tag{14}$$

In terms of (12), then we have

$$f(x_{k+1}) \leq f(x_k) - \frac{1}{2}t_k||\nabla f(x_k)||_2^2 \tag{15}$$

Subtracted both sides with p^*

$$f(x_{k+1}) - p^* \leq f(x_k) - p^* - \frac{1}{2}t_k||\nabla f(x_k)||_2^2 \tag{16}$$

Combined with previously derived inequality

$$||\nabla f(x_k)||_2^2 \geq 2m\big(f(x) - p^*\big) \tag{17}$$

Then we have

$$f(x_{k+1}) - p^* \leq (1 - mt_k)(f(x_k) - p^*) \tag{18}$$

Since $m > 0$ and $t_k > 0$, then

$$f(x_{k+1}) - p^* \leq c(f(x_k) - p^*) \tag{19}$$

where $c = (1 - mt_k) < 1$. And let us apply this formula recursively and get

$$f(x_{k+1}) - p^* \leq c^k(f(x_0) - p^*) \tag{20}$$

which guarantee the convergence of $f(x)$ since $c < 1$. Hence, it can be concluded that with the infinity decreasing step size sequence, the gradient descent must converge to the global optimal solution. □

4 Convex Functions

(a) $f(x) = \sup_i f_i(x)$

Let a and b be two arbitrary distinct points such that $a \in domf$ and $b \in domf$. And let $\lambda \in [0, 1]$.

$$
\begin{aligned}
f(\lambda a + (1-\lambda)b) &= \sup_i f_i(\lambda a + (1-\lambda)b) && (21)\\
&\leq \sup_i \lambda f_i(a) + (1-\lambda) f_i(b) && (22)\\
&\leq \sup_i \lambda f_i(a) + \sup_i (1-\lambda) f_i(b) && (23)\\
&= \lambda \sup_i f_i(a) + (1-\lambda) \sup_i f_i(b) && (24)\\
&= \lambda f(a) + (1-\lambda) f(b) && (25)
\end{aligned}
$$

Since $f(\lambda a + (1-\lambda)b) \leq \lambda f(a) + (1-\lambda)f(b)$ hold for arbitrary distinct points a and b, we can conclude that

$$f_i(x) \text{ is convex } \forall i \Rightarrow f(x) = \sup_i f_i(x) \text{ is convex.} \tag{26}$$

(b) $\lambda_{max}(M)$

Let M_1 and M_2 be arbitrarily distinct matrix of the same dimensionality. Let v_0, v_1, v_2 be eigen vector of $\alpha M_1 + (1-\alpha)M_2$, M_1 and M_2 respectively. and let λ_0, λ_1, λ_2 be the largest eigen value of $\alpha M_1 + (1-\alpha)M_2$, M_1 and M_2 respectively. $\alpha \in [0, 1]$. By eigenvalue decomposition, we have

$$
\begin{aligned}
v_1^T M_1 v_1 &= \lambda_1 && (27)\\
v_2^T M_2 v_2 &= \lambda_2 && (28)\\
v_0^T \big(\alpha M_1 + (1-\alpha)M_2\big) v_0 &= \lambda_0 && (29)
\end{aligned}
$$

It is obvious that

$$
\begin{aligned}
v_1^T M_1 v_1 &= v_0^T M_1 v_0 && (30)\\
v_2^T M_2 v_2 &= v_0^T M_2 v_0 && (31)
\end{aligned}
$$

Otherwise, λ_0, λ_2, λ_2 cannot be the largest eigen value.
Then we have

$$
\begin{aligned}
\alpha v_1^T M_1 v_1 &\geq \alpha v_0^T M_1 v_0 && (32)\\
(1-\alpha) v_2^T M_2 v_2 &\geq (1-\alpha) v_0^T M_2 v_0 && (33)
\end{aligned}
$$

Add above two inequalities up, we have

$$\alpha v_1^T M_1 v_1 + (1-\alpha) v_2^T M_2 v_2 \geq \alpha v_0^T M_1 v_0 + (1-\alpha) v_0^T M_2 v_0 \tag{34}$$

That is

$$\alpha \lambda_1 + (1-\alpha)\lambda_2 \geq v_0^T (\alpha M_1 + (1-\alpha)M_2) v_0 = \lambda_0 \tag{35}$$

Hence, we can conclude that function λ_{max} is convex.

Remark 2. *The eigenvalue of largest magnitude is not convex.*

(c) Weighted shortest path from a to b

For a fixed graph topology with distinct weights w_1 and w_2, we have

$$\lambda f(w_1) + (1-\lambda) f(w_2) = f(\lambda w_1) + f((1-\lambda)w_2) \tag{36}$$

where λ is abitrary real value between 0 and 1. The above step is valid because if we multiply every weight of edge with a constant (λ in this case), then the path between two nodes with minimal weights stay invariant and hence minimal weights of that path becomes a constant (λ) multiple of its original value.

$$f(\lambda w_1) + f((1-\lambda)w_2) \leq f(\lambda w_1 + (1-\lambda)w_2) \tag{37}$$

To simplify the explanation, we denote the cost of minimal-weight path returned by $f(\lambda w_1 + (1-\lambda)w_2)$ under weight w as $PC(w)$). That is to say,

$$f(\lambda w_1 + (1-\lambda)w_2) = PC(\lambda w_1) + PC((1-\lambda)w_2) \tag{38}$$

The justification is as follows:

- If $f(\lambda w_1)$ and $f((1-\lambda)w_2)$ return different paths between two nodes, then by definition we have $f(\lambda w_1) \leq PC(\lambda w_1)$ and $f((1-\lambda)w_2) \leq PC((1-\lambda)w_2)$. Sum up these two inequalities derives $f(\lambda w_1) + f((1-\lambda)w_2) < f(\lambda w_1 + (1-\lambda)w_2)$
- If $f(\lambda w_1)$ and $f((1-\lambda)w_2)$ return the same path between two nodes, it is obvious that the equality holds.

Hence, we have

$$\lambda f(w_1) + (1-\lambda) f(w_2) \leq f(\lambda w_1 + (1-\lambda)w_2) \tag{39}$$

which indicates that f is concave function of w.

5 Convex Functions: Jensen's Inequality

(a) $epi(f)$ is also convex if $f(x)$ is convex

Let (a_1, b_1) and (a_2, b_2) are two distinct pair that belongs to $epi(f)$. That is

$$a_1 \neq a_2, b_1 \neq b_2 \tag{40}$$
$$(a_1, b_1) \in epi(x) \tag{41}$$
$$(a_2, b_2) \in epi(x) \tag{42}$$

We want to prove for arbitrary $\lambda \in [0, 1]$,

$$\lambda(a_1, b_1) + (1 - \lambda)(a_2, b_2) \in epi(x) \tag{43}$$

holds.
In terms of (41) and (42), we have

$$b_1 \geq f(a_1) \tag{44}$$
$$b_2 \geq f(a_2) \tag{45}$$

Multiply both sides with λ and $(1 - \lambda)$ respectively

$$\lambda b_1 \geq \lambda f(a_1) \tag{46}$$
$$(1 - \lambda)b_2 \geq (1 - \lambda)f(a_2) \tag{47}$$

Then add up two terms above, we have

$$\lambda b_1 + (1 - \lambda)b_2 \geq \lambda f(a_1) + (1 - \lambda)f(a_2) \tag{48}$$

Since $f(x)$ is convex, then we have

$$\lambda f(a_1) + (1 - \lambda)f(a_2) \geq f(a_1 + (1 - \lambda)a_2) \tag{49}$$

Then

$$\lambda b_1 + (1 - \lambda)b_2 \geq \lambda f(a_1 + (1 - \lambda)a_2) \tag{50}$$

That is to say,

$$(\lambda a_1 + (1 - \lambda)a_2, \lambda b_1 + (1 - \lambda)b_2) \in epi(f) \tag{51}$$

which can be written as

$$\lambda(a_1, b_1) + (1 - \lambda)(a_2, b_2) \in epi(f) \tag{52}$$

Hence, we can conclude that $epi(f)$ is a convex set.

(b) Finite version of Jensen's inequality

We prove this by induction on m.
Base Case: $m = 1$. It is obvious that $\mathbb{E}(f(x_1)) = f(E(x_1)) = f(x_1)$.
Inductive Cases: assume that for $\sum_{i=1}^{m} p_i = 1$

$$\mathbb{E}[f(X_m)] = \sum_{i=1}^{m} p_i f(x_i) \leq f(\sum_{i=1}^{m} p_i x_i) = f(\mathbb{E}(X_m)) \tag{53}$$

holds and show that for $\sum_{i=1}^{m+1} p_i = 1$,

$$\mathbb{E}[f(X_{m+1})] = \sum_{i=1}^{m+1} p_i f(x_i) \leq f(\sum_{i=1}^{m+1} p_i x_i) = f(\mathbb{E}(X_{m+1})) \tag{54}$$

is true.

Proof.

$$\begin{aligned}
\mathbb{E}[f(X_{m+1})] &= \sum_{i=1}^{m} p_i f(x_i) + p_{m+1} f(x_i) && (55)\\
&= \big(\sum_{i=1}^{m} p_i\big) \sum_{i=1}^{m} \frac{p_i}{\sum_{i=1}^{m} p_i} f(x_i) + p_{m+1} f(x_{m+1}) && (56)\\
&\leq \big(\sum_{i=1}^{m} p_i\big) f(\sum_{i=1}^{m} \frac{p_i}{\sum_{i=1}^{m} p_i} x_i) + p_{m+1} f(x_{m+1}) && (57)\\
&\leq f\big((\sum_{i=1}^{m} p_i) \sum_{i=1}^{m} \frac{p_i}{\sum_{i=1}^{m} p_i} x_i + p_{m+1} x_{m+1}\big) && (58)\\
&= f\big(\sum_{i=1}^{m} p_i x_i + p_{m+1} x_{m+1}\big) && (59)\\
&= f(\sum_{i=1}^{m+1} p_i x_i) && (60)\\
&= f(\mathbb{E}(X_{m+1})) && (61)
\end{aligned}$$

Hence, it is proved that $\mathbb{E}[f(X_{m+1})] \leq f(\mathbb{E}(X_{m+1}))$. And the induction holds. □

From the base case and inductive cases, it can be concluded by induction that

$$\mathbb{E}[f(X)] \leq f(\mathbb{E}(X)) \tag{62}$$

6 Projection

We start by manipulating the solution.
By the definition of projection, we have

$$
\begin{aligned}
x^{(k+1)} &= Proj_{\chi}(x^{(k)} - t_k \nabla f(x^{(k)})) && (63)\\
&= argmin_{x \in \chi} ||x - (x^{(k)} - t_k \nabla f(x^{(k)}))||_2 && (64)\\
&= argmin_{x \in \chi} ||(x - x^{(k)}) + t_k \nabla f(x^{(k)}))||_2^2 && (65)\\
&= argmin_{x \in \chi} \big(||x - x^{(k)}||_2^2 + 2t_k(x - x^{(k)})\nabla f(x^{(k)}) + t_k^2 ||\nabla f(x^{(k)})||_2^2\big) && (66)\\
&= argmin_{x \in \chi} \big(||x - x^{(k)}||_2^2 + 2t_k x \nabla f(x^{(k)}) - 2t_k x^{(k)} \nabla f(x^{(k)}) + t_k^2 ||\nabla f(x^{(k)})||_2^2\big) && (67)
\end{aligned}
$$

Obviously, both $-2t_k x^{(k)} \nabla f(x^{(k)})$ and $t_k^2 ||\nabla f(x^{(k)})||_2^2$ are constant term at step k. Then, we can remove them in the optimization objective.

$$
\begin{aligned}
x^{(k+1)} &= Proj_{\chi}(x^{(k)} - t_k \nabla f(x^{(k)})) && (68)\\
&= argmin_{x \in \chi} \big(||x - x^{(k)}||_2^2 + 2t_k x \nabla f(x^{(k)})\big) && (69)\\
&= argmin_{x \in \chi} \big(2t_k \langle x, \nabla f(x^{(k)})\rangle + ||x - x^{(k)}||_2^2\big) && (70)\\
&= argmin_{x \in \chi} \big(\langle x, \nabla f(x^{(k)})\rangle + \frac{1}{2t_k}||x - x^{(k)}||_2^2\big) && (71)
\end{aligned}
$$

Note that the last step is valid because t_k is constant at step k.
Hence, we proved that

$$
\begin{aligned}
& x^{(k+1)} = argmin_{x \in \chi} \big(\langle x, \nabla f(x^{(k)})\rangle + \frac{1}{2t_k}||x - x^{(k)}||_2^2\big) && (72)\\
\iff & x^{(k+1)} = Proj_{\chi}(x^{(k)} - t_k \nabla f(x^{(k)})) && (73)
\end{aligned}
$$

7 Computing Projections

(a) $\chi = \{x : L_i \leq x_i \leq U_i, i = 1, ..., n\}$

$$argmin_{L_i \leq x_i \leq U_i} ||x_i - z_i||_2^2 \tag{74}$$

Solving this objective, we get

$$x_i^* = max(min(z_i, U_i), L_i),\ \forall i \tag{75}$$

(b) $\chi = \mathbb{R}_+^n$

The solution is

$$x_i^* = max(z_i, L_i),\ \forall i \tag{76}$$

Note that we can simply view $\mathbb{R}_+^n$ as rectangle with $L = 0$ and $U = \infty$.

(c) Euclidean ball: $\{x : ||x||_2 \leq 1\}$

As to the Euclidean ball: $\{x : ||x||_2 \leq 1\}$, we have projection task as follows:

$$argmin_{||x||_2^2 \leq 1} ||x_i - z_i||_2^2 \tag{77}$$

Solving this by using lagrangian

$$L(x, \lambda) = ||x - z||_2 + \lambda(||x||_2^2 - 1) \tag{78}$$

And we get the stationary point by setting gradient to zero,

$$x = \frac{z}{1 + \lambda} \tag{79}$$

Now we need to discuss the value of x and z. If $||z|| \leq 1$, then $\lambda = 0$. But if $||z|| > 1$, then $||x||_2 = \frac{z}{||z||_2}$.

(d) 1-norm ball: $\{x : \Sigma_i |x_i| \leq 1\}$

(e) Positive semidefinite cone: $S_+^n = \{M \in S^n : x^T M x \geq 0, \forall x \in R^n\}$

(f) Probability Simplex: $\chi = \{\Sigma_i x_i = 1, x_i \geq 0, i = 1, ..., n\}$

A Codes Printout

(a) 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
```

(b) 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
```

(c) 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
```

(d) 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
```

(e) 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
```