Homework 2
Shengchao Liu

1. $x_k = \frac{1}{k^k}$

   Because $\lim_{k\to\infty} \frac{x_{k+1}}{x_k} = \frac{k^k}{(k+1)^{k+1}} = \frac{1}{k} = 0$, it is Q-superlinear.

   And $\lim_{k\to\infty} \frac{x_{k+1}}{x_k} = \frac{k^{2k}}{(k+1)^{k+1}} \approx k^{k-1}$, not bounded, not Q-quadradic.

2. $f(x) = \frac{1}{2}x^T Ax$, $\bigtriangledown f(x) = Ax$.

   For a strongly convex f, which satisfies Goldstein, first we may as well find its one-dimensional minimizer α^*.

   $x^T Ap + \alpha^* p^T Ap = 0$, $\alpha^* = -\frac{x^T Ap}{p^T Ap}$.

   Put this α^* back to $f(x + \alpha p)$, we can have:

   $f(x + \alpha^* p) = \frac{1}{2}x^T Ax + \alpha^* x^T Ap + \frac{(\alpha^*)^2}{2} p^T Ap = \frac{1}{2}x^T Ax + \frac{\alpha^*}{2} x^T Ap = \frac{1}{2}x^T Ax + \frac{\alpha^*}{2} \bigtriangledown f^T p$

   Therefore we can get:

   $\frac{1}{2}x^T Ax + (1-c) \bigtriangledown f^T p \le \frac{1}{2}x^T Ax + \frac{\alpha^*}{2} \bigtriangledown f^T p \le \frac{1}{2}x^T Ax + c \bigtriangledown f^T p$, where $c \in \left(0, \frac{1}{2}\right)$.

3. Suppose we set $\phi(\alpha) = a\alpha^2 + b\alpha + c$, then

   $\phi(0) = c$

   $\phi'(0) = b$

   $\phi(\alpha_0) = a\alpha_0^2 + b\alpha_0 + c \Longrightarrow a\alpha_0^2 + \phi'(0)\alpha_0 + \phi(0) \Longrightarrow a = \frac{\phi(\alpha_0) - \phi(0) - \alpha_0\phi'(0)}{\alpha_0^2}$

   $\therefore \phi(\alpha) = \frac{\phi(\alpha_0) - \phi(0) - \alpha_0\phi'(0)}{\alpha_0^2}\alpha^2 + \phi'(0)\alpha + \phi(0)$.

   And because α_0 does not s.t. sufficient decrease condition, which means $f(x + \alpha_0) > f(x) + c_1\alpha_0 \bigtriangledown f^T p$.

   Thus we can get $\phi(\alpha_0) > \phi(0) + c_1\alpha_0\phi'(0)$.

   $\Longrightarrow \phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0 > (c_1 - 1)\alpha_0\phi'(0)$

   $\Longrightarrow -(\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0) < (1 - c_1)\alpha_0\phi'(0)$

   Because $1 - c_1 > 0$, and $\alpha_0 > 0$ and $\phi'(0) < 0$, thus we know both sides are smaller than 0.

   $\Longrightarrow -\frac{\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0}{\phi'(0)} > (1 - c_1)\alpha_0$

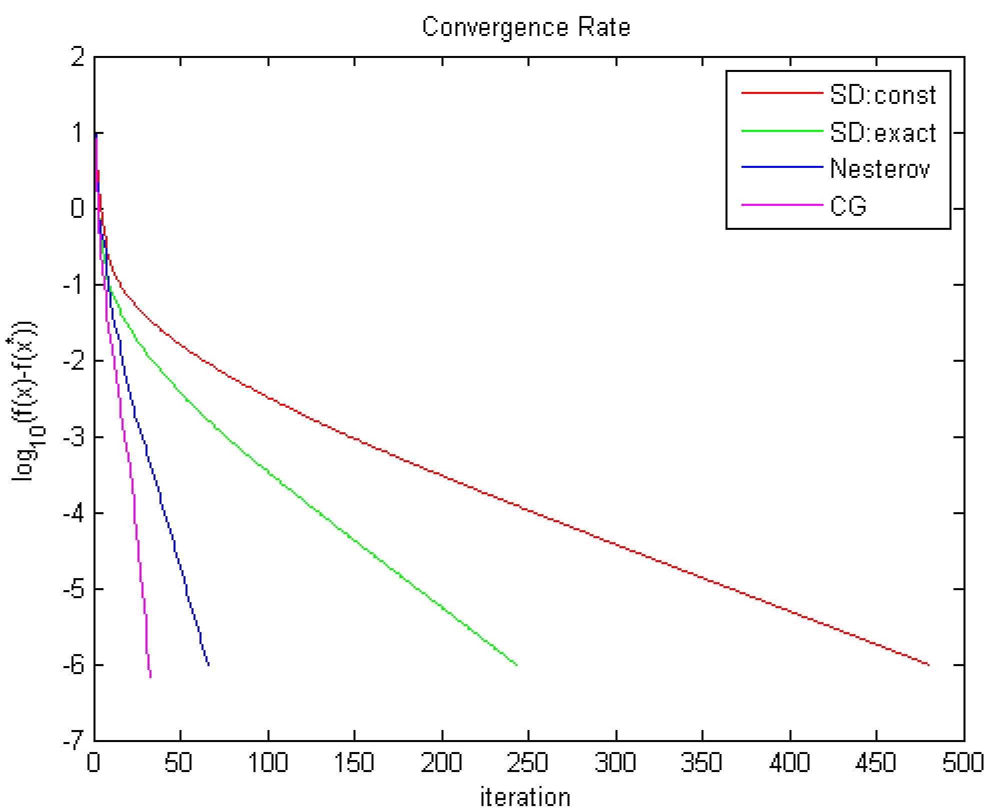
   $\Longrightarrow -\frac{\phi'(0)\alpha_0^2}{2(\phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0)} < \frac{\alpha_0}{2(1 - c_1)}$

   $\Longrightarrow \alpha_1 < \frac{\alpha_0}{2(1 - c_1)}$

4. (a)

	sd	sde	nest	cg
1	388	196	58	32
2	413	211	57	32
3	441	221	57	31
4	344	175	55	30
5	384	195	58	32
6	379	192	57	31
7	427	216	56	32
8	422	213	59	32
9	479	243	66	33
10	480	243	66	33
average	415.7	210.5	58.9	31.8

(b) For the last run, we can get this plot:



(c) Yes, this result satisfies match with what we proved before.

From the plotting, we can tell that steepest descent with fixed step converges linearly. $k \le \frac{L}{m} \log\left((f(x) - f^*)/\epsilon\right) \approx 1.5e3$

And exact line search, which takes more computation complexity, but less iterations, also converges linearly.

And Nestrov converges in R-linearly faster than two showed before, while conjugate converges fastest.

Codes are below:

```
%% global variable
mu=0.01; L=1; kappa=L/mu;
n=100;
A=randn(n,n); [Q,R] = qr(A);
D=rand(n,1); D=10.^D; Dmin=min(D); Dmax=max(D);
D=(D-Dmin)/(Dmax-Dmin);
D=mu+D*(L-mu);
A=Q'*diag(D)*Q;

epoch_step  = 30000;
trial_num = 10;
av_sd_list=ones(trial_num,1)*epoch_step;
av_sde_list=ones(trial_num,1)*epoch_step;
av_nest_list=ones(trial_num,1)*epoch_step;
av_cg_list=ones(trial_num,1)*epoch_step;

for step = 1:trial_num
    x0=randn(n,1);%use a different x0 for each trial

    %% Steepest descent with sd \alpha
    x_sd_list = zeros(n, epoch_step);
    x_sd_list(:,1)=x0;

    for k = 1:epoch_step-1
        alpha = 1.0 / L;
        x_k = x_sd_list(:,k);
        if 0.5 * x_k' * A * x_k <= 10^(-6)
            av_sd_list(step)=k;
            break
        end
        delta_k = A * x_k;
        x_sd_list(:,k+1) = x_k - alpha * delta_k;
    end


    %% Steepest descent with exact line search
    x_sde_list = zeros(n, epoch_step);
    x_sde_list(:,1)=x0;

    for k = 1:epoch_step-1
        x_k = x_sde_list(:,k);
        if 0.5 * x_k' * A * x_k <= 10^(-6)
            av_sde_list(step)=k;
```

```
            break
        end
        delta_k = A * x_k;
        alpha = (delta_k' * delta_k) / (delta_k' * A * delta_k) ;
        x_sde_list(:,k+1) = x_k - alpha * delta_k;
    end


    %% Nesterov's method
    x_nest_list = zeros(n, epoch_step);
    x_nest_list(:,1)=x0;

    m=mu;
    alpha=1.0/L;
    beta=(sqrt(kappa)-1)/(sqrt(kappa)+1);

    for k = 1:epoch_step-1
        x_k = x_nest_list(:,k);
        if 0.5 * x_k' * A * x_k <= 10^(-6)
            av_nest_list(step)=k;
            break
        end
        if k == 1
            y_k = x_k ;
        else
            y_k = x_k + beta * (x_k - x_nest_list(:,k-1));
        end

        delta_k = A * y_k;
        x_nest_list(:,k+1) = y_k - alpha * delta_k;
    end

    %% Conjugate gradient
    x_cg_list = zeros(n, epoch_step);
    x_cg_list(:,1)=x0;

    r = A * x0;
    p = -r;

    for k = 1:epoch_step-1
        x_k = x_cg_list(:,k);
        if 0.5 * x_k' * A * x_k < 10^(-6)
            av_cg_list(step)=k;
            break;
```

```
        end
        alpha = (r' * r) / (p' * A * p);
        x_cg_list(:,k+1) = x_k + alpha * p;
        r_prev = r;
        r = r + alpha * A * p;
        beta = r' * r / (r_prev' * r_prev);
        p = -r + beta * p;
    end


end

av_sd = mean(av_sd_list);
av_sde = mean(av_sde_list);
av_nest = mean(av_nest_list);
av_cg = mean(av_cg_list);
table = [[1:10]', av_sd_list, av_sde_list, av_nest_list, av_cg_list];

fprintf(1, ' steepest descend -fixed steps : %7.1f\n', av_sd);
fprintf(1, ' steepest descend -exact steps : %7.1f\n', av_sde);
fprintf(1, ' Nestrov : %7.1f\n', av_nest);
fprintf(1, ' conjugate gradient : %7.1f\n', av_cg);
```

And the output is:

```
 steepest descend -fixed steps :   415.7
 steepest descend -exact steps :   210.5
 Nestrov :    58.9
 conjugate gradient :    31.8
```