Convex and Nonsmooth Optimization
HW6

Sébastien Kleff - sk8001

03/13/2020

My commented MATLAB codes are available in appendix.

Outline

We report the results for 3 problems :

- Section 1 : Quadratic objective with Hilbert matrix from HW5
- Section 2 : BV 9.30 (p.519)
- Section 3 : Nesterov's worst-case example (p.67) for $M = 100$
- Section 4 : Nesterov's worst-case example (p.67) for $M = 10000$

The MATLAB codes used to generate the plots and numerical results of each section are available in Appendices A, B, C respectively. Each of these 3 problems is solved using 3 methods :

- Nesterov method (accelerated gradient)
- Gradient descent with fixed step size $t = \frac{1}{M}$
- Gradient descent with fixed step size $t = \frac{2}{m+M}$

The MATLAB functions for Nesterov's optimal gradient and the classical gradient descent with fixed step size are available in Appendices D, E.

1 Hilbert

The MATLAB code corresponding to this section is available in Appendix A. We solve the optimization problem

$$\min_x f(x) = \frac{1}{2}x^T Ax + b^T x \tag{1}$$

Since A is positive definite, the minimizer x^* and optimal value p^* of (1) are known

$$\begin{aligned} x^* &= -A^{-1}b \\ p^* &= -\frac{1}{2}b^T A^{-1}b \end{aligned}$$

This problem is solved using the 3 aforementioned methods. The maximum of number or iterations is set to 100, the tolerance to 10^{-6} and the starting point to $x^{(0)} = 1 \in \mathbb{R}^{100}$. The bounds m and M are set as the minimum and maximum eigenvalues of the Hessian A and the condition number is denoted $\kappa = \frac{M}{m}$. As observed in HW5, this problem is ill-conditioned :

$$\begin{aligned} m &\simeq 3.2879e-06 \\ M &\simeq 1.5671 \\ \kappa &\simeq 4.7661e+05 \end{aligned}$$

1.1 Nesterov's optimal gradient

We define

$$q = \frac{1-\sqrt{1\kappa}}{1+\sqrt{1/\kappa}} \simeq 0.9971$$

According to the theoretical results seen in class, Nesterov's sequence is lower bounded as

$$f(x^{(k)}) - p^* \geq \frac{m}{2}q^{2k}\|x^{(0)} - x^*\|^2 \tag{2}$$

Figure 1 shows the objective error $f(x^{(k)}) - p^*$ in a log plot and the theoretical lower bound. We can see that inequality (2) is verified. The algorithm doesn't converge, in a sense that it stops after the maximum number of iterations is reached.

1.2 Gradient descent with step $t = \frac{1}{M}$

According to the theoretical results seen in class, the upper bound on the objective error for fixed step gradient descent with $t = \frac{1}{M}$ is given by

$$f(x^{(k)}) - p^* \leq \left(1 - \frac{1}{\kappa}\right)^k (f(x^{(0)}) - p^*) \tag{3}$$

Figure 2 shows the objective error ratio $\frac{f(x^{(k)})-p^*}{f(x^{(0)})-p^*}$ in a log plot and the theoretical upper bound. We can see that inequality (3) is verified.

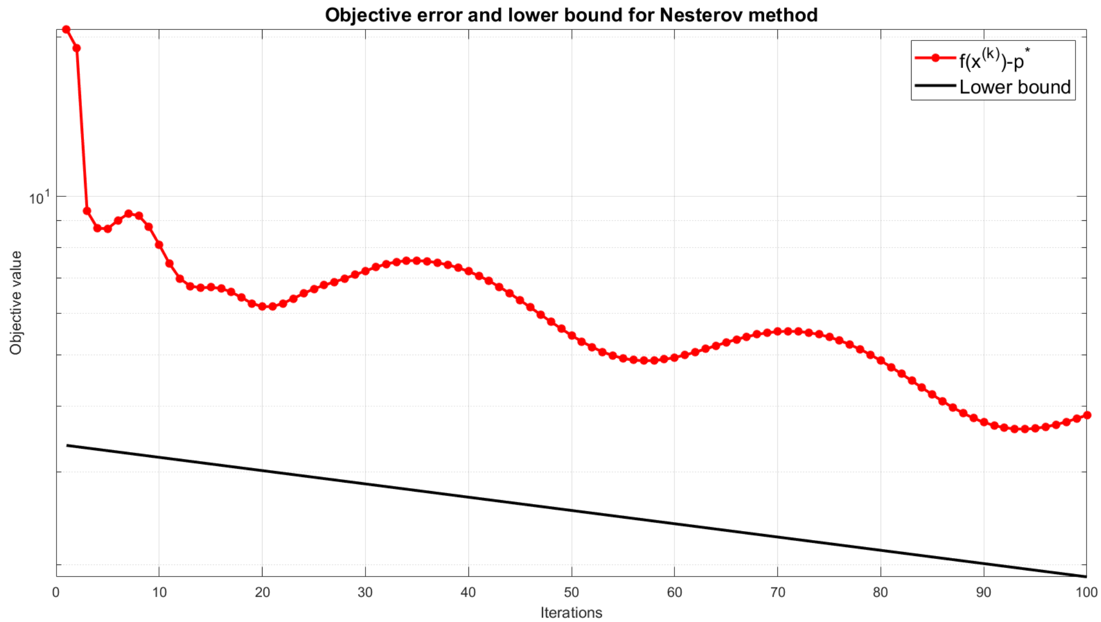


Figure 1: Log plot of the error on the quadratic objective (1) minimized using Nesterov's optimal gradient method and its lower bound

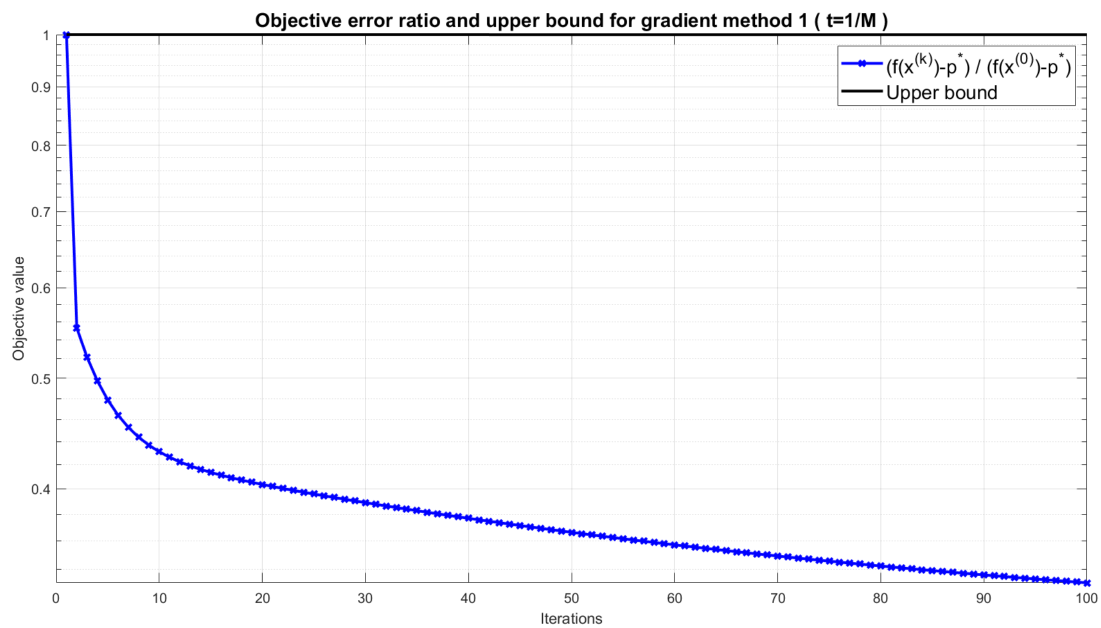


Figure 2: Log plot of the error ratio on the quadratic objective (1) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

1.3 Gradient descent with step $t = \frac{2}{m+M}$

According to the theoretical results seen in class, the upper bound on the objective error for fixed step gradient descent with $t = \frac{2}{m+M}$ is given by

$$f(x^{(k)}) - p^* \leq \kappa \left(\frac{1 - 1/\kappa}{1 + 1/\kappa} \right)^{2k} (f(x^{(0)}) - p^*) \tag{4}$$

Figure 3 shows the objective error ratio $\frac{f(x^{(k)})-p^*}{f(x^{(0)})-p^*}$ in a log plot and the theoretical upper bound. We can see that inequality (4) is verified.

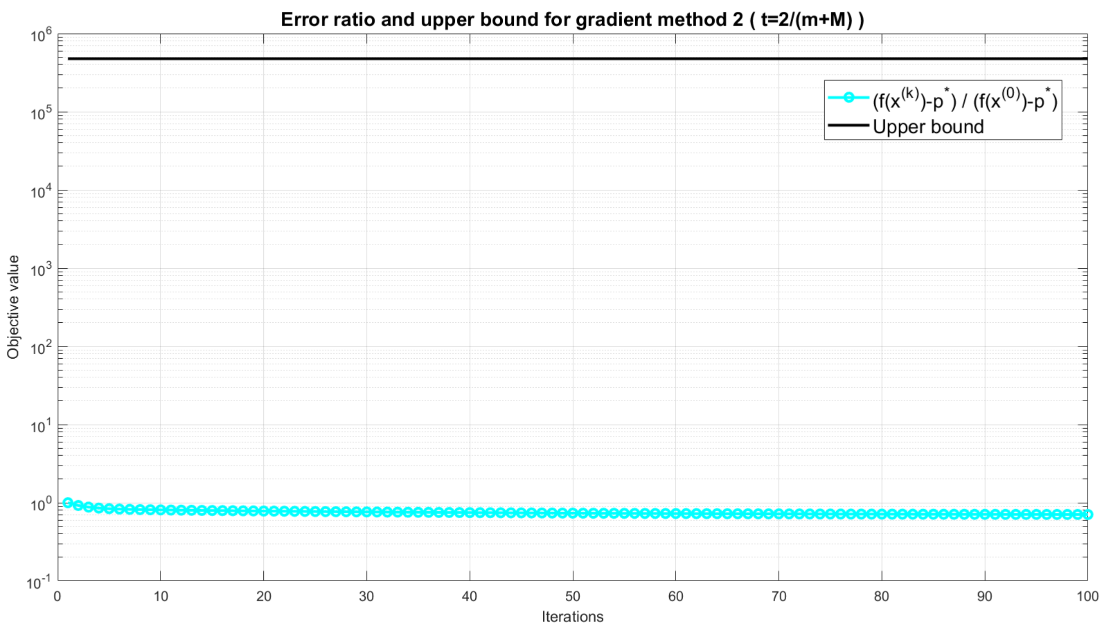
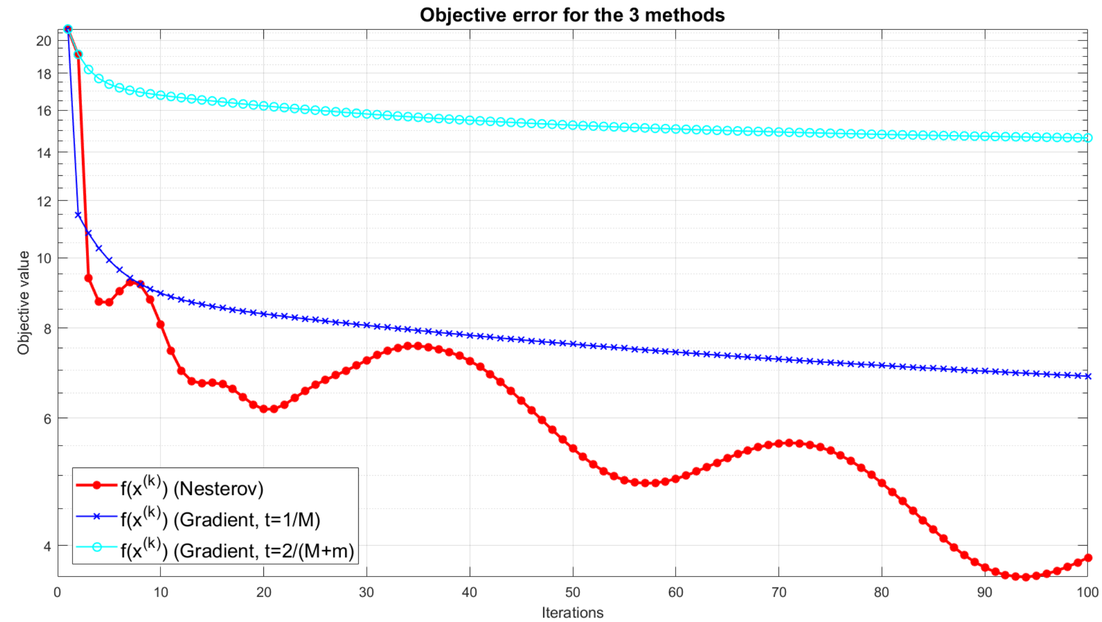


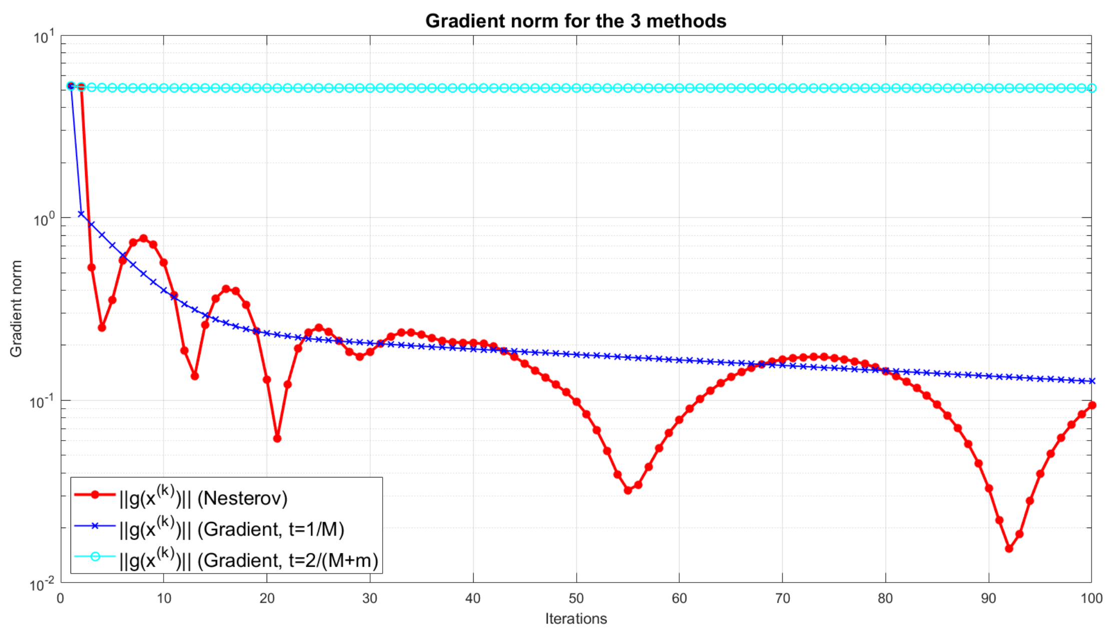
Figure 3: Log plot of the error ratio on the quadratic objective (1) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound

1.4 Discussion

For the 3 methods, the convergence is very slow because of the high condition number. However it can be seen that Nesterov's optimal gradient performs overall better than the two gradient descents. Figures 4a and 4b show the error and gradient norm for all 3 methods in log plots. In particular, Nesterov's method converges much faster than gradient descent with step $t = \frac{2}{m+M}$ because of its the term $\sqrt{\kappa}$ (in q) instead of κ. This gets even more visible if we increase the number of iterations (see Figure 5).



(a) Log plot of the errors on the quadratic objective (1) minimized by the 3 methods.



(b) Log plot of the gradient norm on the quadratic objective (1) minimized by the 3 methods.

Figure 4: Hilbert

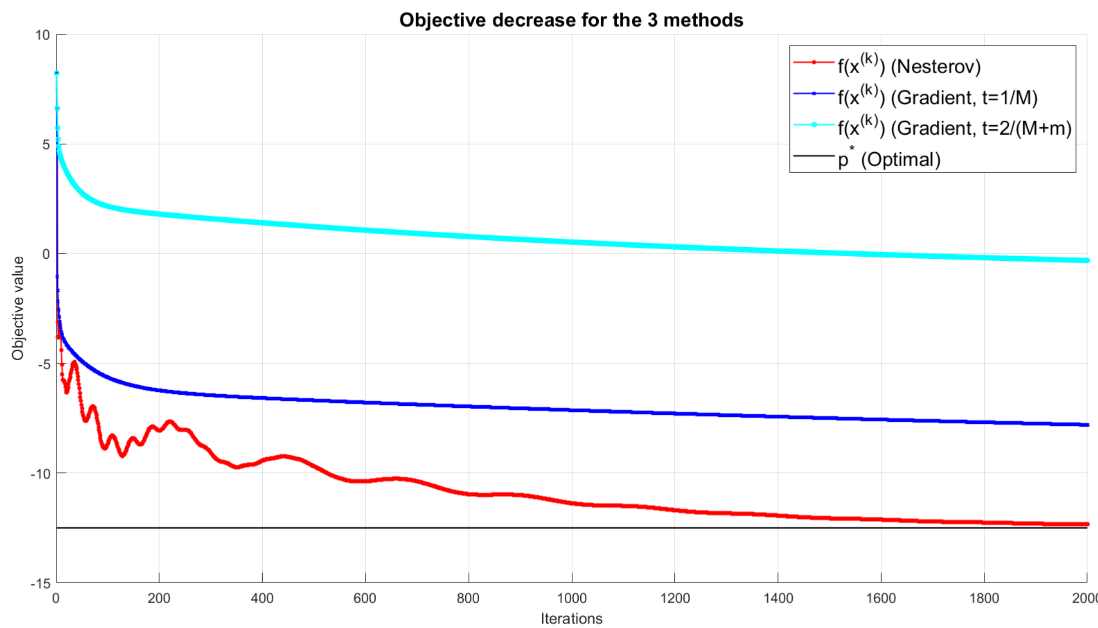


Figure 5: Errors on the quadratic objective (1) minimized by the 3 methods for 2000 iterations.

2 BV 9.30

We consider now the problem BV 9.30 (p.519). Since we don't have access to uniform bounds m and M on the Hessian over the whole space, we need to estimate them by sampling eigenvalues of the Hessian at different points. For instance, we can take as a first guess $M = \lambda_{max}(\nabla^2 f(x^{(0)}))$ and then perform a gradient descent with fixed step size $t = \frac{1}{M}$. Then, we simply evaluate the Hessian along the resulting sequence $x^{(0)}, x^{(1)}, ...$ and estimate m and M by the minimum of the smallest eigenvalues and the maximum of the largest eigenvalues respectively :

$$
\begin{aligned}
m &= \min_k \lambda_{min}(\nabla^2 f(x^{(k)})) \simeq 2.0000 \\
M &= \max_k \lambda_{max}(\nabla^2 f(x^{(k)})) \simeq 281.5852 \\
\kappa &= \frac{M}{m} \simeq 140.7926
\end{aligned}
$$

These bounds may be loose over parts of domain, but lacking a better estimation method we will use them in the following.

2.1 Nesterov's optimal gradient

No closed-form solution is available so we estimate p^* by the last element of the sequence resulting from Nesterov's optimal gradient

$$p^* \simeq -67.4637$$

Figure 6 shows the error on the objective on a log plot along with the lower bound (2). Once again we can observe that the lower bound property is satisfied.

2.2 Gradient descent with step $t = \frac{1}{M}$

Figure 7 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

2.3 Gradient descent with step $t = \frac{2}{m+M}$

Figure 8 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

2.4 Discussion

Figures 9a and 9b show the error and gradient norm for all 3 methods in log plots. We observe that Nesterov's method converges faster to the optimal value.

Figure 6: Log plot of the error for the objective BV 9.30 minimized using Nesterov's optimal gradient method and its lower bound

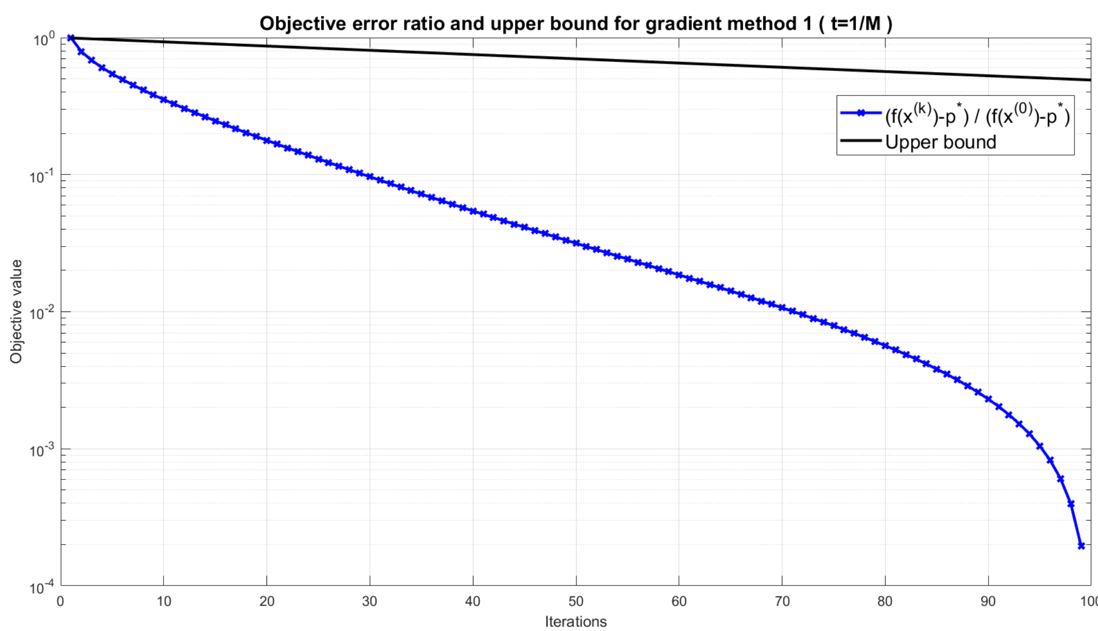


Figure 7: Log plot of the error for the objective BV 9.30 minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

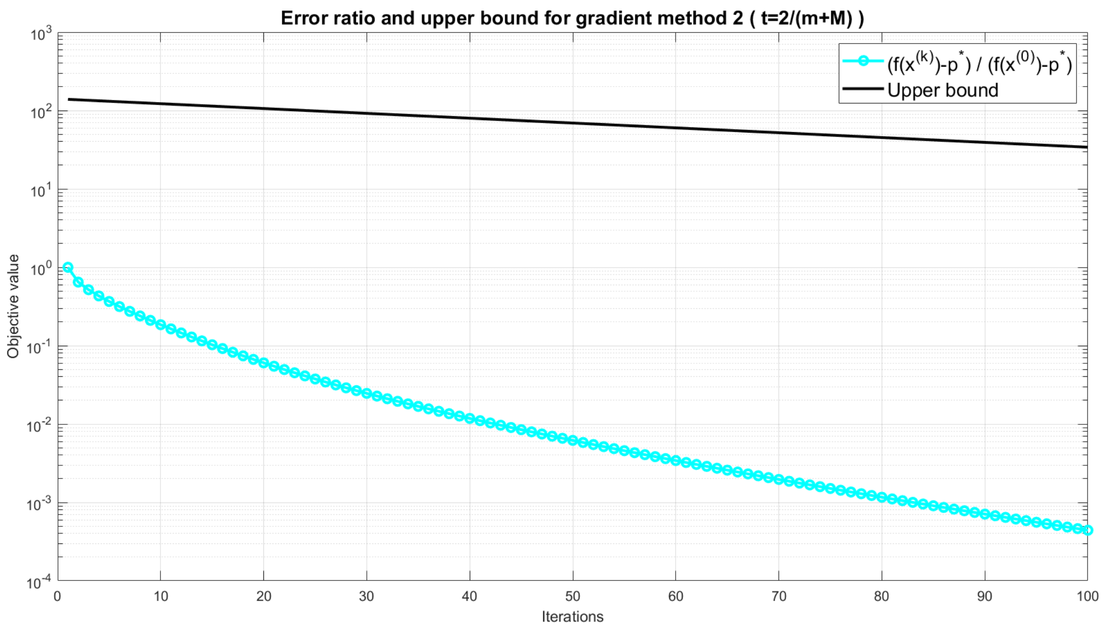
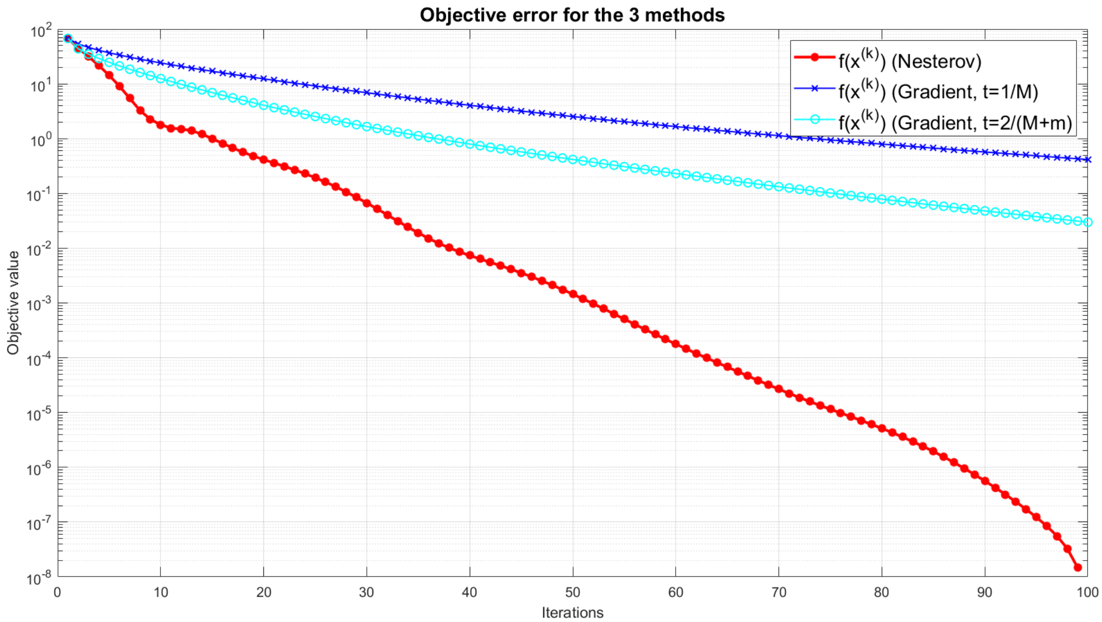
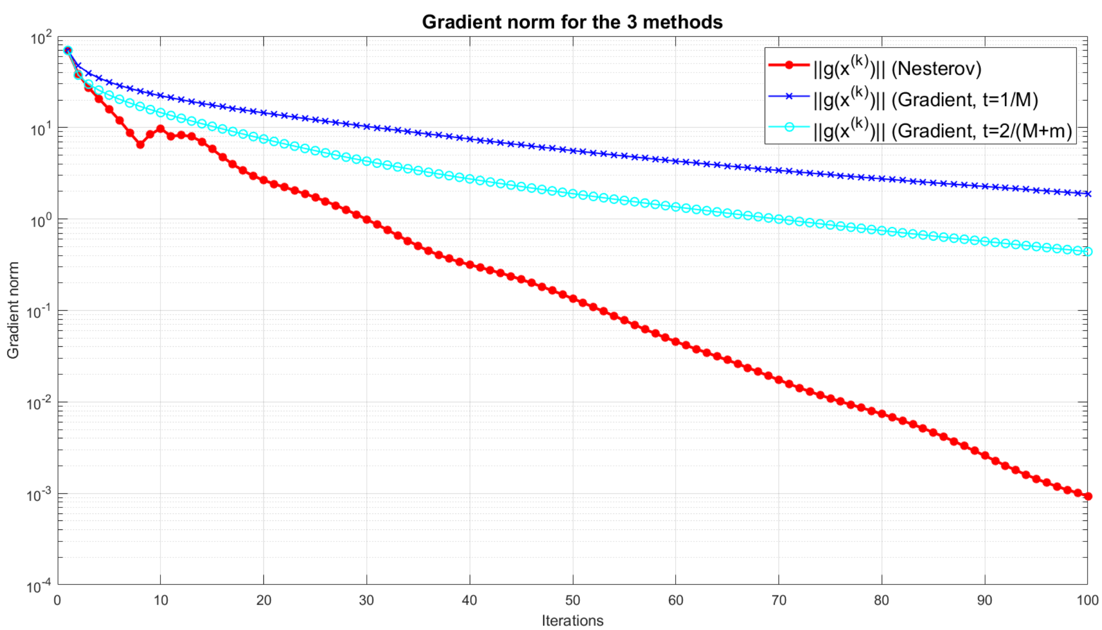


Figure 8: Log plot of the error for the objective BV 9.30 minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound



(a) Log plot of the errors for BV 9.30 minimized by the 3 methods.



(b) Log plot of the gradient norm for BV 9.30 minimized by the 3 methods.

Figure 9: BV 9.30

3 Nesterov's example (M=100)

We consider now Nesterov's worst case example (p.67).

3.1 Nesterov's optimal gradient

We calculate the optimizer x^* the recursive formula from the notes, derived by setting the gradient of the objective function to 0. For $j = 3...n$, $x_{j+1} = q^{j+1}$. Then for x_1 and x_2, knowing $x_3 = q^3$, we solve

$$x_2 - 2\frac{M+m}{M-m}x_1 + 1 = 0$$
$$x_3 - 2\frac{M+m}{M-m}x_2 + x_1 = 0$$

We get $p^* = f(x^*)$ for $M = 100$

$$p^* \simeq -10.125$$

Figure 10 shows the error on the objective on a log plot along with the lower bound (2). Once again we can observe that the lower bound property is satisfied.

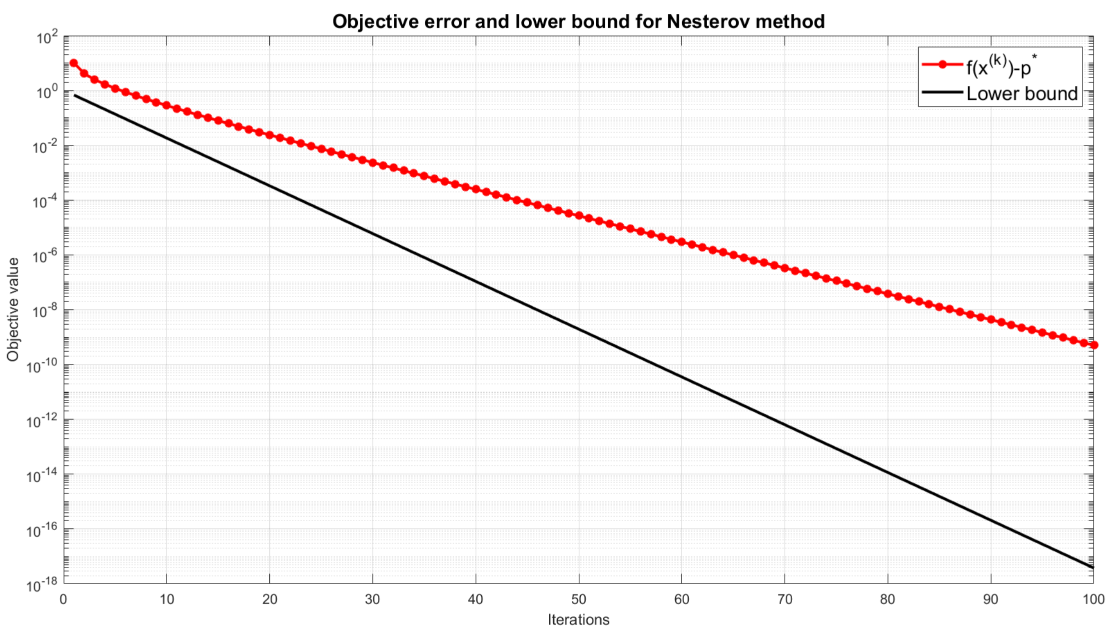


Figure 10: Log plot of the error for Nesterov's example ($M = 100$) minimized using Nesterov's optimal gradient method and its lower bound

3.2 Gradient descent with step $t = \frac{1}{M}$

Figure 11 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

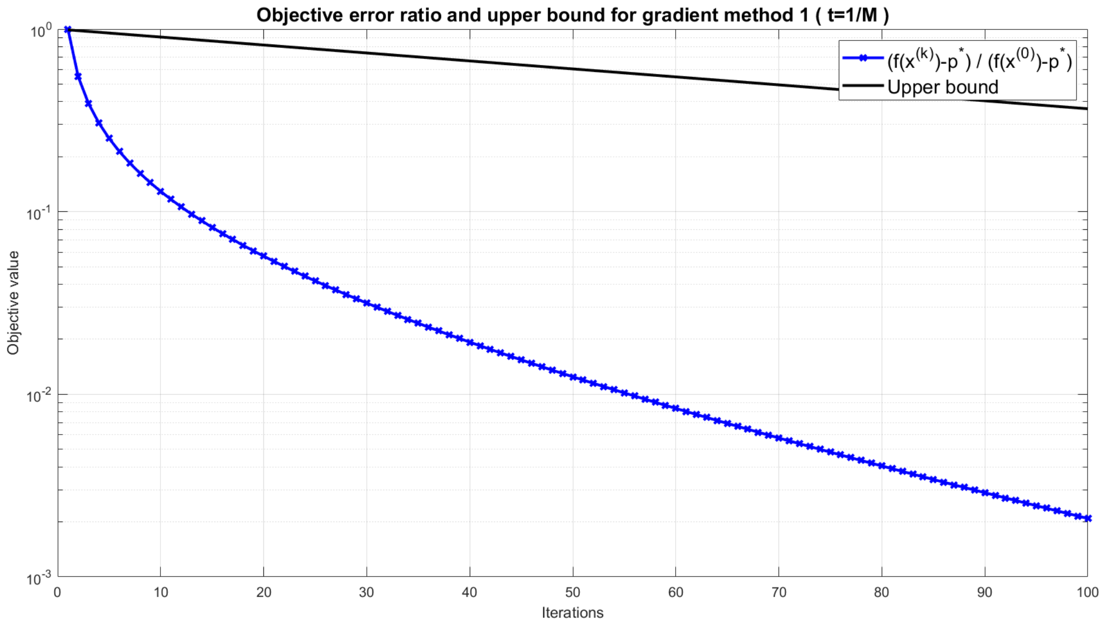


Figure 11: Log plot of the error for Nesterov's example ($M = 100$) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

3.3 Gradient descent with step $t = \frac{2}{m+M}$

Figure 12 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

3.4 Discussion

Figures 13a and 13b show the objective error and gradient norm for the 3 methods in log plots. As expected, Nesterov's method is more efficient that the fixed-step gradient descent with $t = \frac{2}{M+m}$, which itself outperforms $t = \frac{1}{M}$.

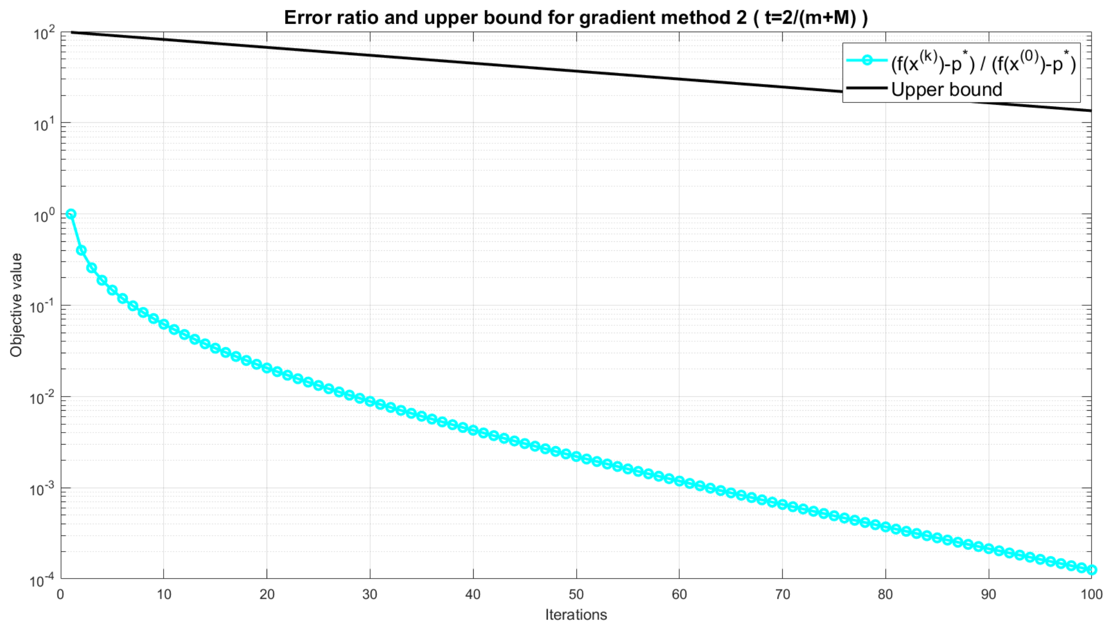
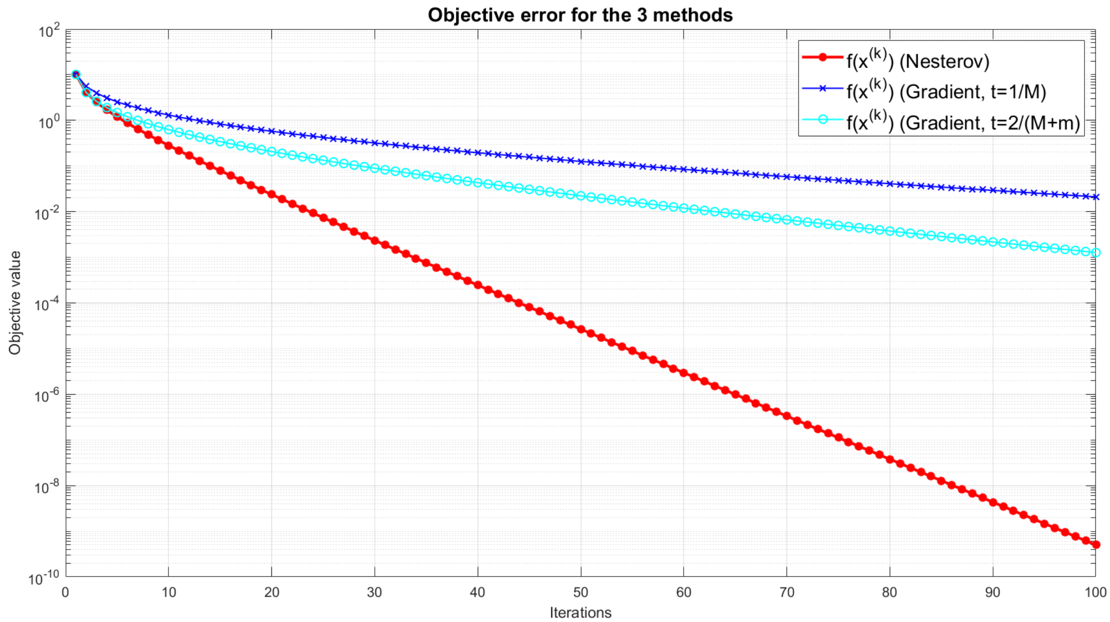
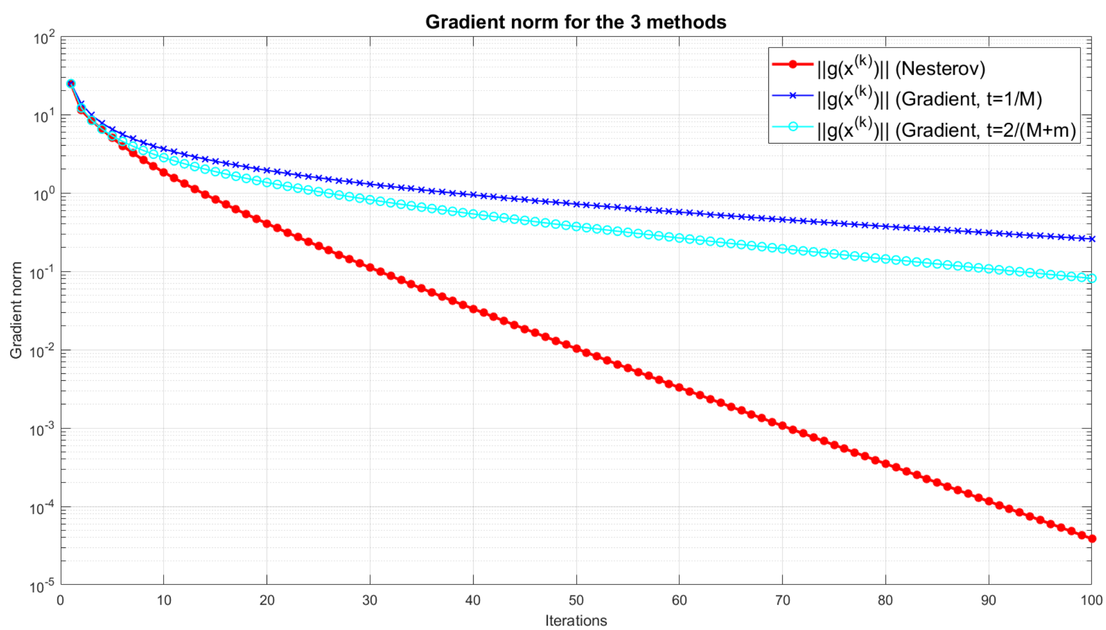


Figure 12: Log plot of the error for Nesterov's example ($M = 100$) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound



(a) Log plot of the errors for Nesterov's example ($M = 100$) minimized by the 3 methods.



(b) Log plot of the gradient norm for Nesterov's example ($M = 100$) minimized by the 3 methods.

Figure 13: Nesterov's example for $M = 100$

4 Nesterov's example (M=10000)

Solving the same system as in the previous section we get for $M = 10000$

$$p^* \simeq -1225.125$$

4.1 Nesterov's optimal gradient

Figure 14 shows the error on the objective on a log plot along with the lower bound (2). Once again we can observe that the lower bound property is satisfied.

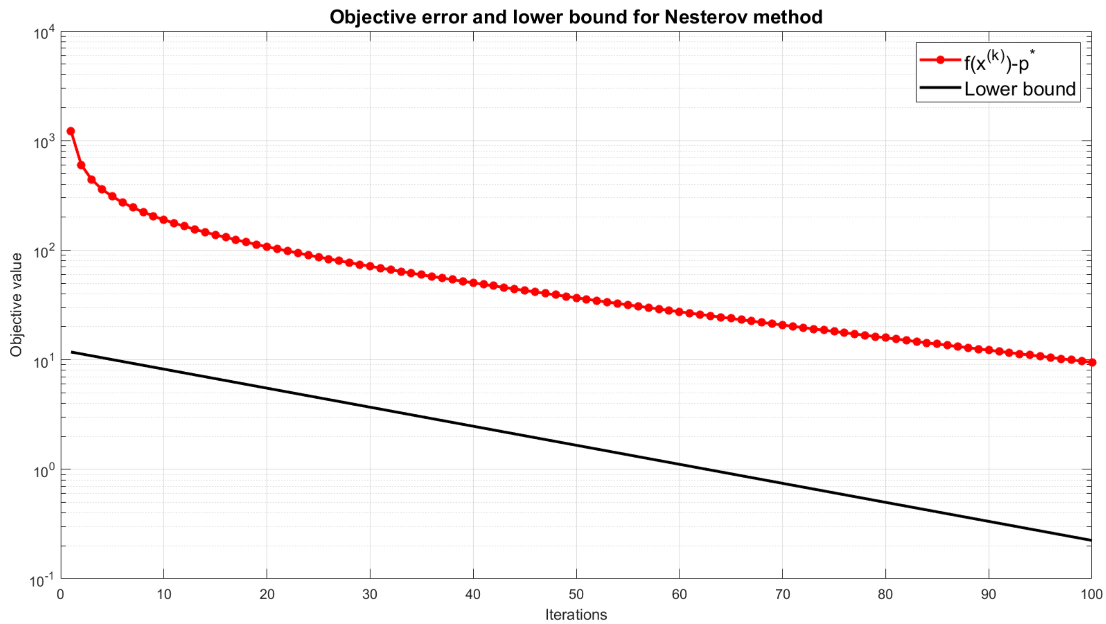


Figure 14: Log plot of the error for Nesterov's example ($M = 10000$) minimized using Nesterov's optimal gradient method and its lower bound

4.2 Gradient descent with step $t = \frac{1}{M}$

Figure 15 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

4.3 Gradient descent with step $t = \frac{2}{m+M}$

Figure 16 shows the error ratio on the objective on a log plot along with the upper bound (2). As in the previous problem, we can see that the upper bound property is satisfied.

4.4 Discussion

Figures 17a and 17b show the objective error and gradient norm for the 3 methods in log plots. As before, Nesterov's method outperforms the two fixed-step gradient descents, but the convergence is much slower than in Section 3 since the condition number is higher ($M = 10000$ instead of $M = 100$). This

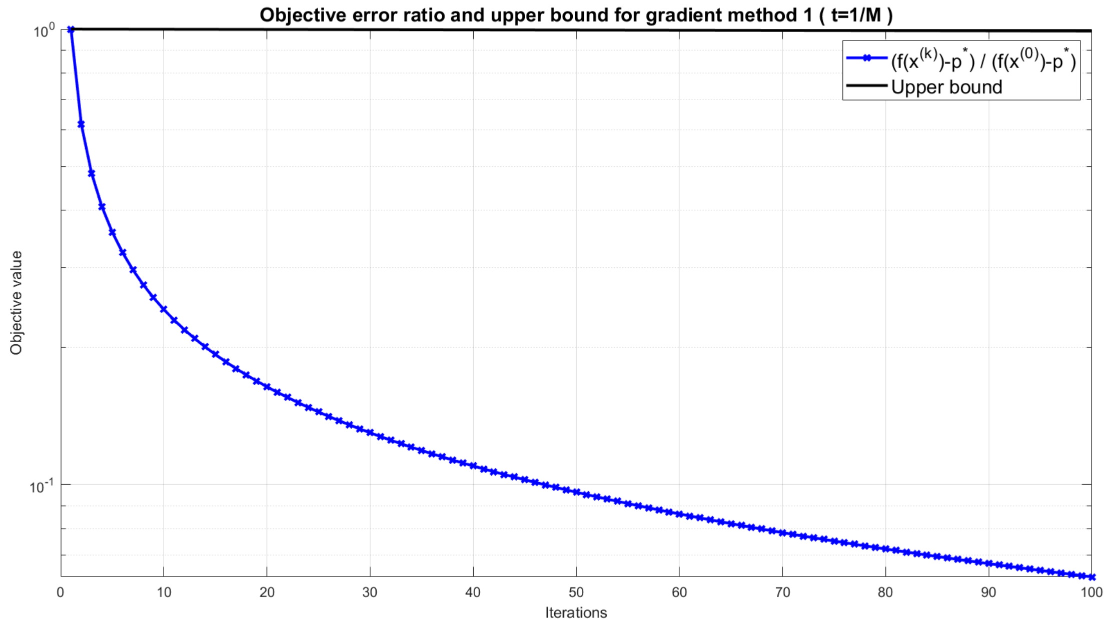


Figure 15: Log plot of the error for Nesterov's example ($M = 10000$) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

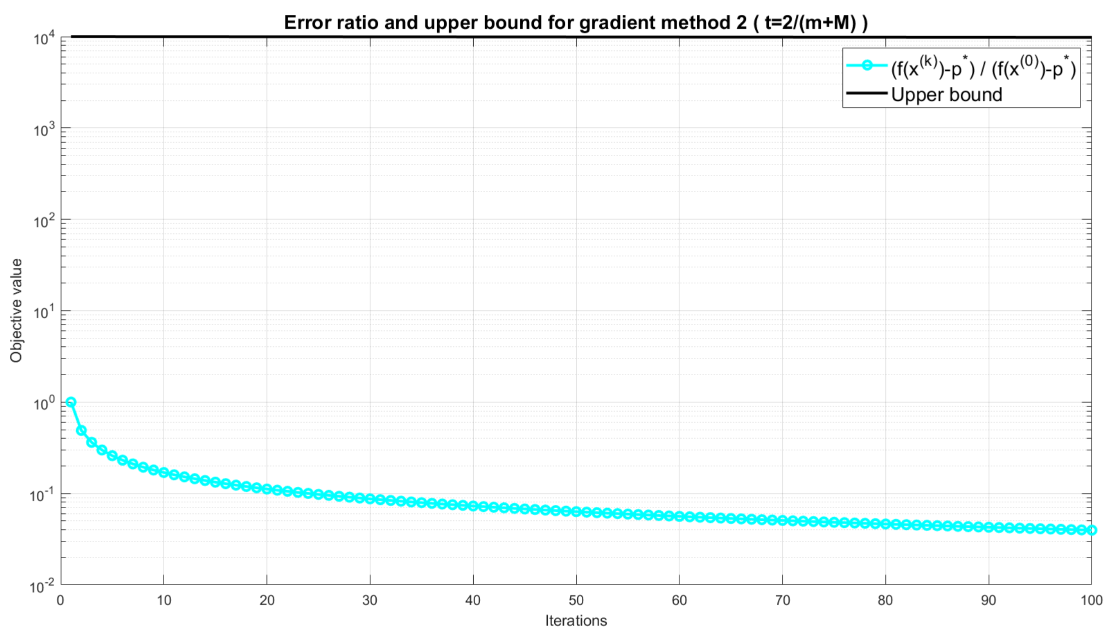
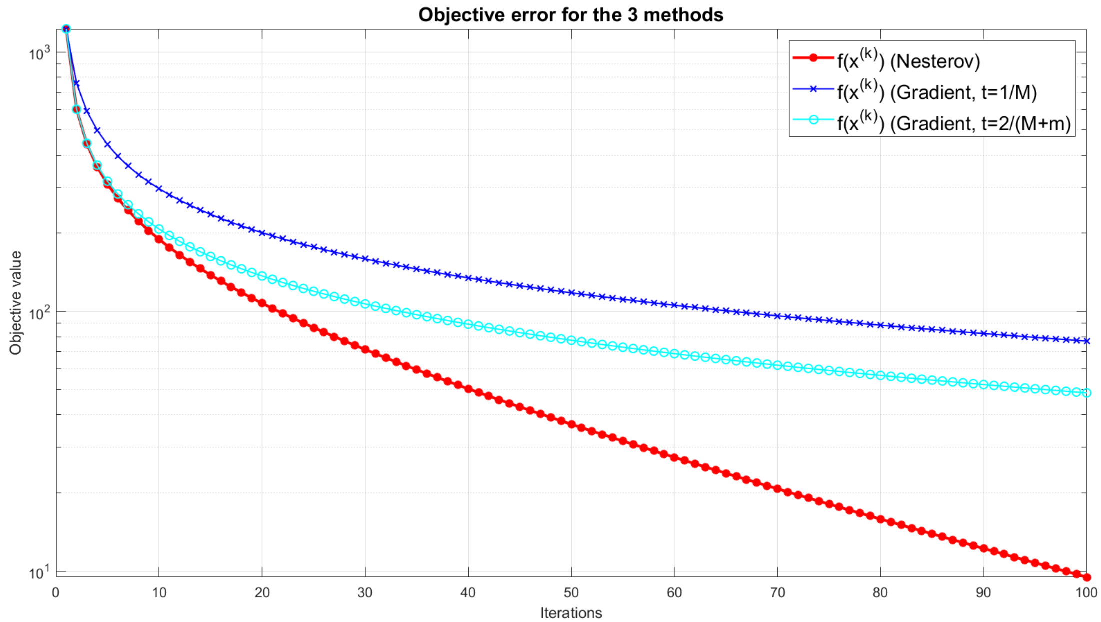
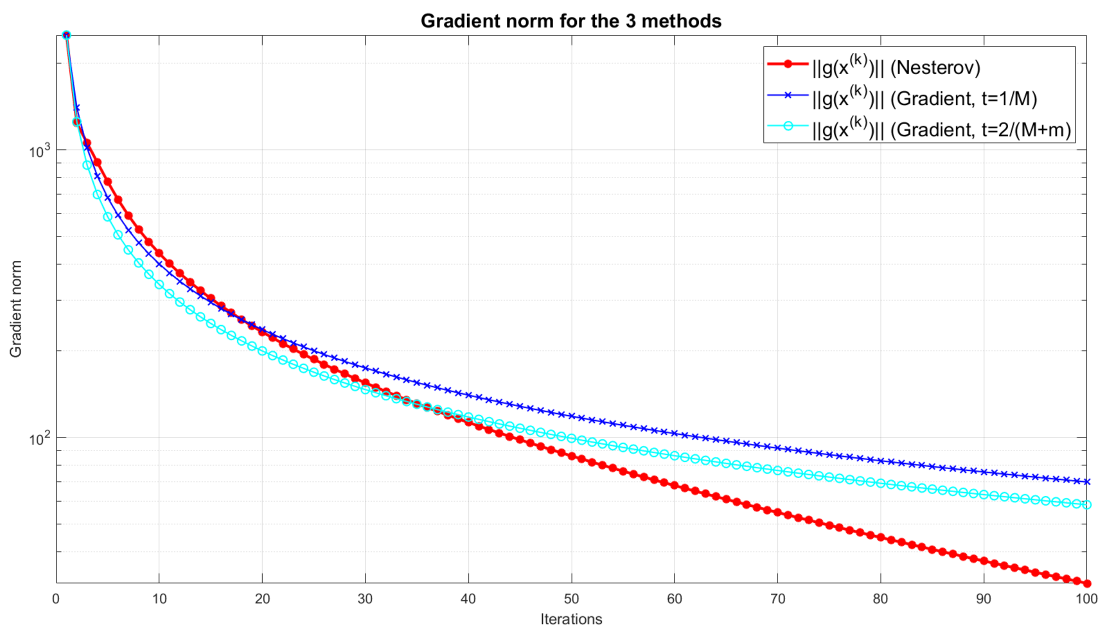


Figure 16: Log plot of the error for Nesterov's example ($M = 10000$) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound

indeed affects the convergence rate in all 3 methods as shown in the notes, or in Section 1 (it makes it closer to 1).



(a) Log plot of the errors for Nesterov's example ($M = 10000$) minimized by the 3 methods.



(b) Log plot of the gradient norm for Nesterov's example ($M = 10000$) minimized by the 3 methods.

Figure 17: Nesterov's example for $M = 10000$

A MATLAB code for Hilbert (Section 1)

```matlab
function [f, g, h, x] = hilbert_fun(x,A,b)
% quadratic function f(x) = 1/2 x'Ax + b'x
% returns value, gradient and hessian and x

% INPUT : 'x' : current point
%         'A' : quadratic term
%         'b' : linear term

% OUPTUT : 'f' : objective function value at x
%          'g' : gradient at x
%          'h' : hessian at x
%          'x' : x

f = 0.5*x'*A*x + b'*x;
g = A*x + b;
h = A;
```

```matlab
%% HW 6 - Quadratic problem of HW5 (Hilbert matrix)

% We define an ill-conditioned matrix A and some vector b
n = 5;
A = hilb(n);
b = ones(n,1);

% The quadratic objective function is defined in quad.m
fun = @(x)hilbert_fun(x, A, b);

% Closed-form solution to the quadratic optimization problem
xstar = A\-b;
pstar = fun(xstar);

% Condition number of A
M = max(eig(A));
m = min(eig(A));
kappa = M/m;

% Starting point & algo parameters (tolerance, #iterations)
x0 = ones(n,1);
tol = 1e-6;
maxit = 2000; %100;

% Iteration counter for plots (x-axis)
t = linspace(1, maxit, maxit);

%% Nesterov's Optimal Gradient (accelerated gradient)
% Parameter q
q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );

% Launch Nesterov algorithm
[f_all_nest, gnorm_all_nest] = nesterov(fun, x0, M, q, tol, maxit);

% Verify the lower bound of Nesterov
```

```matlab
% Plot objective error and lower bound in a LOG plot
figure
    % Plot error
err_nest = f_all_nest - pstar;
semilogy(err_nest, 'r-*','LineWidth',1,'MarkerSize',3);
hold on
    % Plot lower bound
lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
semilogy(lower_bd_nest, 'k-', 'LineWidth', 1);
    % Custom
grid on
title('Objective error and lower bound for Nesterov method',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)})-p^{*}', ...
       'Lower bound', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Gradient descent with fixed step size t = 1/M
% Step size
t1 = 1/M;

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);

% Verify convergence
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
semilogy(err_ratio_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
hold on
    % Plot upper bound
upper_bd_grad1 = (1-1/kappa).^t;
semilogy(upper_bd_grad1, 'k-', 'LineWidth', 1);
    % Custom
grid on
title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Gradient descent with fixed step size = t = 2/(M+m)
% Step size
t2 = 2/(M+m);

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);

% Verify convergence properties
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad2 = ( f_all_grad2 - pstar ) / (f_all_grad2(1) - pstar);
```

```
semilogy(err_ratio_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
hold on
    % Plot upper bound
upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^(2*t);
semilogy(upper_bd_grad2, 'k-', 'LineWidth', 1);
    % Custom
grid on
title('Error ratio and upper bound for gradient method 2 ( t=2/(m+M) )',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');


%% Plot objective for 3 methods in same plot
% Objective
figure, hold on, grid on
% Nesterov
plot(t, f_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
% Gradient 1
plot(t, f_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
% Gradient 2
plot(t, f_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
% Optimal
plot(t, pstar*ones(1,maxit),'k-','LineWidth',1);
% Legend, axis and title
title('Objective decrease for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)}) (Nesterov)', ...
       'f(x^{(k)}) (Gradient, t=1/M)', ...
       'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
       'p^{*} (Optimal)', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Plot gradient norm for 3 methods in same plot
% Gradient norm
figure, hold on, grid on
% Nesterov
plot(t, gnorm_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
% Gradient 1
plot(t, gnorm_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
% Gradient 2
plot(t, gnorm_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
% Legend, axis and title
title('Gradient norms for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Gradient norm');
legend('||g(x^{(k)})|| (Nesterov)', ...
       '||g(x^{(k)})|| (Gradient, t=1/M)', ...
       '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Plot objective error for 3 methods in same LOG plot
```

```
% Objective log error
figure
% Nesterov
semilogy(t, f_all_nest - pstar, 'r-*','LineWidth',1,'MarkerSize',3);
hold on
% Gradient 1
semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1,'MarkerSize',3);
% Gradient 2
semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1,'MarkerSize',3);
% Legend, axis and title
title('Objective error for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)}) (Nesterov)', ...
       'f(x^{(k)}) (Gradient, t=1/M)', ...
       'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on

%% Plot gradient norm for 3 methods in same LOG plot
% Gradient norm log plot
figure
% Nesterov
semilogy(t, gnorm_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
hold on
% Gradient 1
semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
% Gradient 2
semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
% Legend, axis and title
title('Gradient norm for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Gradient norm');
legend('||g(x^{(k)})|| (Nesterov)', ...
       '||g(x^{(k)})|| (Gradient, t=1/M)', ...
       '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on
```

B MATLAB code for BV 9.30 (Section 2)

```
function [f, g, h, x] = bv930_fun(x, a)
% Objective function of Ex 9.30 (BV p.519)
% This function returns the evaluation, gradient and hessian and x

% INPUT : 'x' : current point
%         'a' : matrix a

% OUPTUT : 'f' : objective function value at x
%          'g' : gradient at x
%          'h' : hessian at x
%          'x' : x
```

```
% Sizes
n = size(a,1);
m = size(a,2);

% Compute ai'*x, i=1..m
ax = zeros(m,1);
for i=1:m
    ax(i) = a(:,i)'*x;
end

% If x is not in dom(f) return default values for f and g
if max(ax)≥1 || max(abs(x))≥1
    f = inf;
    g = nan*ones(n,1);

% Else continue
else
%% Calculate f := f(x)
    f=0;
        % First term of f : sum over m terms with ai'*x
    for i=1:m
        f = f - log(1-ax(i));
    end
        % Second term of f : sum over n terms with xi^2
    for i=1:n
        f = f - log(1-x(i)^2);
    end

%% Calculate gradient
    g = zeros(n,1);

    % Analytic formula for the gradient
    for i=1:n
        g(i) = 2*x(i)/(1-x(i)^2);
        for j=1:m
            g(i) = g(i) + a(i,j)/(1-a(:,j)'*x);
        end
    end

    %% Calculate hessian
    h = zeros(n,n);

    % Analytic formula for the hessian
    for i=1:n
        for k=1:n
            if i==k
                h(i,k) = 2*(1 + x(k)^2) / ( (1 - x(k)^2)^2 );
            else
                h(i,k) = 0;
            end
            for j=1:m
                h(i,k) = h(i,k) + a(i,j)*a(k,j)/( (1-a(:,j)'*x)^2 );
            end
        end
    end
```

```

end

end
```

```
%% HW6 - BV 9.30 (p.519)

% Load data set for BV 9.30
A = load('Adata.mat').A;
n = size(A,1);

% The objective function of BV 9.30 is implemented in bv930_fun.m
fun = @(x)bv930_fun(x, A);

% Starting point & algo parameters (tolerance, #iterations)
x0 = zeros(n,1);
tol = 1e-6;
maxit = 100;

% Iteration counter for plots (x-axis)
t = linspace(1, maxit, maxit);

%% Newton's method (ref for pstar)
% % Launch Newton's method to evaluate pstar and bounds on Hessian m,M
% [f_all, gnorm_all, h_all, x_all] = newtmeth(fun, x0, tol, maxit);
% min_eigs = [];
% max_eigs = [];
% for i=1:length(x_all)
%     h = hessian(A, x{i});
%     min_eigs = [min_eigs, min(eig(h_all{i}))];
%     max_eigs = [max_eigs, max(eig(h_all{i}))];
% end
% m = min(min_eigs)
% M = max(max_eigs)
% kappa = M/m

%% Gradient descent to evaluate (m,M)
% Get a 1st guess for m,M with Hessian at starting point
[f, g, h, x] = fun(x0);
m = min(eig(h));
M = max(eig(h));
% Launch grad descent with t=1/M to re-estimate bounds from sequence
[f_all, gnorm_all, x_all] = gradmeth(fun, x0, 1/M, tol, maxit);
min_eigs = [];
max_eigs = [];
for i=1:length(x_all)
    h = hessian(A, x_all{i});
    min_eigs = [min_eigs, min(eig(h))];
    max_eigs = [max_eigs, max(eig(h))];
end
m = min(min_eigs);
M = max(max_eigs);
kappa = M/m;

% Define optimal value to be last element of Newton sequence
pstar = f_all(end);
```

```
xstar = x_all{end};

%% Nesterov's Optimal Gradient (accelerated gradient)
% Parameter q
q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );

% Launch Nesterov algorithm
[f_all_nest, gnorm_all_nest] = nesterov(fun, x0, M, q, tol, maxit);

% Estimate optimal value by last element of Nesterov sequence
pstar = f_all_nest(end);

% Verify the lower bound of Nesterov
% Plot objective error and lower bound in a LOG plot
figure
    % Plot error
err_nest = f_all_nest - pstar;
semilogy(err_nest, 'r-*', 'LineWidth', 2);
hold on
    % Plot lower bound
lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
semilogy(lower_bd_nest, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Objective error and lower bound for Nesterov method',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)})-p^{*}', ...
       'Lower bound', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Gradient descent with fixed step size t = 1/M
% Step size
t1 = 1/M;

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);

% Verify convergence
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
semilogy(err_ratio_grad1, 'b-x', 'LineWidth', 2);
hold on
    % Plot upper bound
upper_bd_grad1 = (1-1/kappa).^t;
semilogy(upper_bd_grad1, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');
```

```
%% Gradient descent with fixed step size = t = 2/(M+m)
% Step size
t2 = 2/(M+m);

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);

% Verify convergence properties
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad2 = ( f_all_grad2 - pstar ) / (f_all_grad2(1) - pstar);
semilogy(err_ratio_grad2, 'c-o', 'LineWidth', 2);
hold on
    % Plot upper bound
upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^t;
semilogy(upper_bd_grad2, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Error ratio and upper bound for gradient method 2 ( t=2/(m+M) )',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');


%% Plot objective for 3 methods in same plot
% Objective
figure, hold on, grid on
% Nesterov
plot(t, f_all_nest, 'r-*','LineWidth',2);
% Gradient 1
plot(t, f_all_grad1, 'b-x','LineWidth',1);
% Gradient 2
plot(t, f_all_grad2, 'c-o','LineWidth',1);
% Optimal
plot(t, pstar*ones(1,maxit),'k-','LineWidth',2);
% Legend, axis and title
title('Objective decrease for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)}) (Nesterov)', ...
       'f(x^{(k)}) (Gradient, t=1/M)', ...
       'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
       'p^{*} (Optimal)', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Plot gradient norm for 3 methods in same plot
% Gradient norm
figure, hold on, grid on
% Nesterov
plot(t, gnorm_all_nest, 'r-*','LineWidth',2);
% Gradient 1
plot(t, gnorm_all_grad1, 'b-x','LineWidth',1);
```

```
% Gradient 2
plot(t, gnorm_all_grad2, 'c-o','LineWidth',1);
% Legend, axis and title
title('Gradient norms for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Gradient norm');
legend('||g(x^{(k)})|| (Nesterov)', ...
       '||g(x^{(k)})|| (Gradient, t=1/M)', ...
       '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Plot objective error for 3 methods in same LOG plot
% Objective log error
figure
% Nesterov
semilogy(t, f_all_nest - pstar, 'r-*','LineWidth',2);
hold on
% Gradient 1
semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1);
% Gradient 2
semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1);
% Legend, axis and title
title('Objective error for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)}) (Nesterov)', ...
       'f(x^{(k)}) (Gradient, t=1/M)', ...
       'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on

%% Plot gradient norm for 3 methods in same LOG plot
% Gradient norm log plot
figure
% Nesterov
semilogy(t, gnorm_all_nest, 'r-*','LineWidth',2);
hold on
% Gradient 1
semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1);
% Gradient 2
semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1);
% Legend, axis and title
title('Gradient norm for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Gradient norm');
legend('||g(x^{(k)})|| (Nesterov)', ...
       '||g(x^{(k)})|| (Gradient, t=1/M)', ...
       '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on
```

C MATLAB code for Nesterov's example (Section 3)

```matlab
function [f, g, x] = nestex_fun(x, m, M)
% Objective function of Nesterov's worst case example (p.67)
% This function returns the evaluation, gradient and hessian and x

% Sizes
n = size(x,1);

% Value of objective
f = ( (M-m)/8 )*( x(1)^2 - 2*x(1) ) + (m/2)*norm(x,2)^2;
for i=1:n-1
    f = f + ( (M-m)/8 )*( x(i)-x(i+1) )^2;
end

% Sparse tridiagonal matrix
% T = sparse(full(gallery('tridiag',n,-1,2,-1)));

% % e1
% e1 = zeros(n,1);
% e1(1) = 1;
%
% % Gradient
% g = sparse(( ((M-m)/4)*T + m*eye(n) )*x - ((M-m)/4)*e1 );
%
T = zeros(n, n);
T(1:1+n:n*n) = 2;
T(n+1:1+n:n*n) = -1;
T(2:1+n:n*n) = -1;
T = sparse(T);

e = zeros(n,1);
e(1) = 1;
g = sparse((((M-m)/4)*T + m*eye(n))*x - ((M-m)/4)*e);
end
```

```matlab
%% HW6 - Nesterov's example (p.67) - M=100 & M=10000

% Init size and bounds
n = 10000;
m = 1;
M = 100; % or M=10000;
kappa = M/m;

% The objective function of BV 9.30 is implemented in bv930_fun.m
fun = @(x)nestex_fun(x, m, M);

% Starting point & algo parameters (tolerance, #iterations)
x0 = zeros(n,1);
size(x0)
tol = 1e-6;
maxit = 100;

% Iteration counter for plots (x-axis)
t = linspace(1, maxit, maxit);

%% Nesterov's Optimal Gradient (accelerated gradient)
% Parameter q
```

```
q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );

% Calculate optimal value (analytical solution taken from lecture notes)
% I solved the linear system for x(1) and x(2) using "\" in symbolic
xtsar = zeros(n,1);
% For the rest of the vector, it's simply x(i) = q^i
for i=3:n
    xstar(i,1) = q^i;
end
tmp = (M + m)/(M-m);
xstar(2,1) = (2*tmp*xstar(3)+1)/(4*tmp^2-1);
xstar(1,1) = (1 + xstar(2))/(2*tmp);
[pstar,gstar,xstar] = fun(xstar);

% Launch Nesterov algorithm
[f_all_nest, gnorm_all_nest, x_all_nest] = nesterov(fun, x0, M, q, tol, maxit);

% Verify the lower bound of Nesterov
% Plot objective error and lower bound in a LOG plot
figure
    % Plot error
err_nest = f_all_nest - pstar;
semilogy(err_nest, 'r-*', 'LineWidth', 2);
hold on
    % Plot lower bound
lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
semilogy(lower_bd_nest, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Objective error and lower bound for Nesterov method',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)})-p^{*}', ...
       'Lower bound', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Gradient descent with fixed step size t = 1/M
% Step size
t1 = 1/M;

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);

% Verify convergence
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
semilogy(err_ratio_grad1, 'b-x', 'LineWidth', 2);
hold on
    % Plot upper bound
upper_bd_grad1 = (1-1/kappa).^t;
semilogy(upper_bd_grad1, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
    'FontSize', 14);
```

```
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');

%% Gradient descent with fixed step size = t = 2/(M+m)
% Step size
t2 = 2/(M+m);

% Launch the gradient descent implemented in gradmeth.m
[f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);

% Verify convergence properties
% Plot objective error ratio and upper bound in a LOG plot
figure
    % Plot error ratio
err_ratio_grad2 = ( f_all_grad2 - pstar ) / (f_all_grad2(1) - pstar);
semilogy(err_ratio_grad2, 'c-o', 'LineWidth', 2);
hold on
    % Plot upper bound
upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^t;
semilogy(upper_bd_grad2, 'k-', 'LineWidth', 2);
    % Custom
grid on
title('Error ratio and upper bound for gradient method 2 ( t=2/(m+M) )',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
       'Upper bound', ...
       'FontSize', 14, 'Location', 'NorthEast');


% %% Plot objective for 3 methods in same plot
% % Objective
% figure, hold on, grid on
% % Nesterov
% plot(t, f_all_nest, 'r-*','LineWidth',2);
% % Gradient 1
% plot(t, f_all_grad1, 'b-x','LineWidth',1);
% % Gradient 2
% plot(t, f_all_grad2, 'c-o','LineWidth',1);
% % Optimal
% plot(t, pstar*ones(1,maxit),'k-','LineWidth',2);
% % Legend, axis and title
% title('Objective decrease for the 3 methods',...
%     'FontSize', 14);
% xlabel('Iterations');
% ylabel('Objective value');
% legend('f(x^{(k)}) (Nesterov)', ...
%        'f(x^{(k)}) (Gradient, t=1/M)', ...
%        'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
%        'p^{*} (Optimal)', ...
%        'FontSize', 14, 'Location', 'NorthEast');

% %% Plot gradient norm for 3 methods in same plot
% % Gradient norm
```

```
% figure, hold on, grid on
% % Nesterov
% plot(t, gnorm_all_nest, 'r-*','LineWidth',2);
% % Gradient 1
% plot(t, gnorm_all_grad1, 'b-x','LineWidth',1);
% % Gradient 2
% plot(t, gnorm_all_grad2, 'c-o','LineWidth',1);
% % Legend, axis and title
% title('Gradient norms for the 3 methods',...
%     'FontSize', 14);
% xlabel('Iterations');
% ylabel('Gradient norm');
% legend('||g(x^{(k)})|| (Nesterov)', ...
%        '||g(x^{(k)})|| (Gradient, t=1/M)', ...
%        '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
%        'FontSize', 14, 'Location', 'NorthEast');

%% Plot objective error for 3 methods in same LOG plot
% Objective log error
figure
% Nesterov
semilogy(t, f_all_nest - pstar, 'r-*','LineWidth',2);
hold on
% Gradient 1
semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1);
% Gradient 2
semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1);
% Legend, axis and title
title('Objective error for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Objective value');
legend('f(x^{(k)}) (Nesterov)', ...
       'f(x^{(k)}) (Gradient, t=1/M)', ...
       'f(x^{(k)}) (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on

%% Plot gradient norm for 3 methods in same LOG plot
% Gradient norm log plot
figure
% Nesterov
semilogy(t, gnorm_all_nest, 'r-*','LineWidth',2);
hold on
% Gradient 1
semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1);
% Gradient 2
semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1);
% Legend, axis and title
title('Gradient norm for the 3 methods',...
    'FontSize', 14);
xlabel('Iterations');
ylabel('Gradient norm');
legend('||g(x^{(k)})|| (Nesterov)', ...
       '||g(x^{(k)})|| (Gradient, t=1/M)', ...
       '||g(x^{(k)})|| (Gradient, t=2/(M+m)', ...
       'FontSize', 14, 'Location', 'NorthEast');
grid on
```

D MATLAB code Nesterov's optimal gradient method

```
function [f_all, gnorm_all, x_all] = nesterov(fun, x0, M, q, tol, maxit)
% Code for Accelerated Gradient Method (Nesterov's Optimal Gradient Method)

% INPUT : 'fun'   : anonymous function
%         'x0'    : starting point
%         'q'     : parameter
%         'tol'   : tolerance threshold on gradient norm (stopping criteria)
%         'maxit  : max number of iterations

% OUPTUT : 'f_all'     : sequence of objective function values
%          'gnorm_all' : sequence of gradient norms

% To store values of x, f(x) and norm(g(x))
f_all = [];
gnorm_all = [];
x_all = {};

% Initialize current xk and yk at the starting point x0
xk = x0;
yk = x0;

% Main loop of the gradient descent
k=1;
while (k ≤ maxit)

    % Evaluate the anonymous function & gradient at current yk
    [f_yk, g_yk] = fun(yk);

    % Record yk and its value
    f_all = [f_all; f_yk];
    gnorm_all = [gnorm_all; norm(g_yk)];
    x_all{end+1} = yk;

    % Stop if we are below tolerance level
    if norm(g_yk) < tol
        break
    end

    % Take the step for x and y
    xkp1 = yk -(1/M)*g_yk;
    ykp1 = xkp1 + q*(xkp1 - xk);
    xk = xkp1;
    yk = ykp1;
    k = k+1;
end
```

E MATLAB code fixed step size gradient descent

```
function [f_all, gnorm_all, x_all] = gradmeth(fun, x0, t, tol, maxit)
% Code for Accelerated Gradient Method (Nesterov's Optimal Gradient Method)
```

```

% INPUT : 'fun'  : anonymous function
%         'x0'   : starting point
%         't'    : fixed step size
%         'tol'  : tolerance threshold on gradient norm (stopping criteria)
%         'maxit : max number of iterations

% OUPTUT : 'f_all'     : sequence of objective function values
%          'gnorm_all' : sequence of gradient norms
%          'x_all'     : sequence of points

% To store values of x, f(x) and norm(g(x))
f_all = [];
gnorm_all = [];
x_all = {};

% Initialize current xk at the starting point x0
xk = x0;

% Main loop of the gradient descent
k=1;
while (k <= maxit)

    % Evaluate the anonymous function & gradient at current xk
    [fk, gk] = fun(xk);

    % Record xk and its value
    f_all = [f_all; fk];
    gnorm_all = [gnorm_all; norm(gk)];
    x_all{end+1} = xk;

    % Stop if we are below tolerance level
    if gnorm_all(k) < tol
        break
    end

    % Compute the descent direction at xk  ( = negative gradient)
    dx = -gk;

    % Take the step
    xk = xk + t*dx;
    k = k+1;
end
```