

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

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

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 of 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

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

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

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

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.

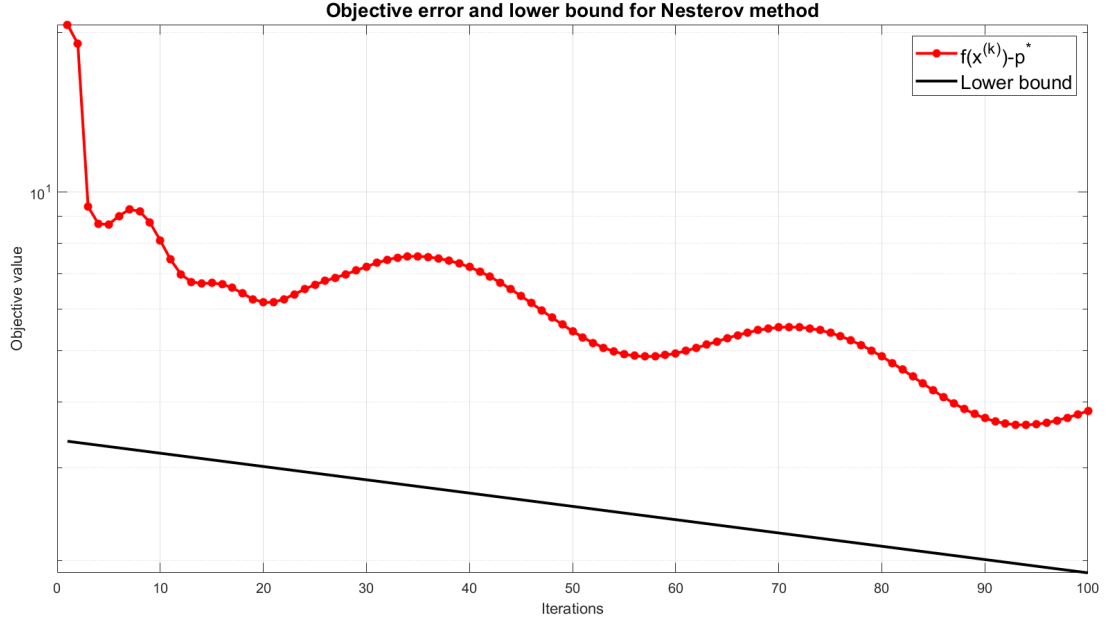


Figure 1: Log plot of the error on the quadratic objective (1) minimized using Nesterov's optimal gradient method and its lower bound

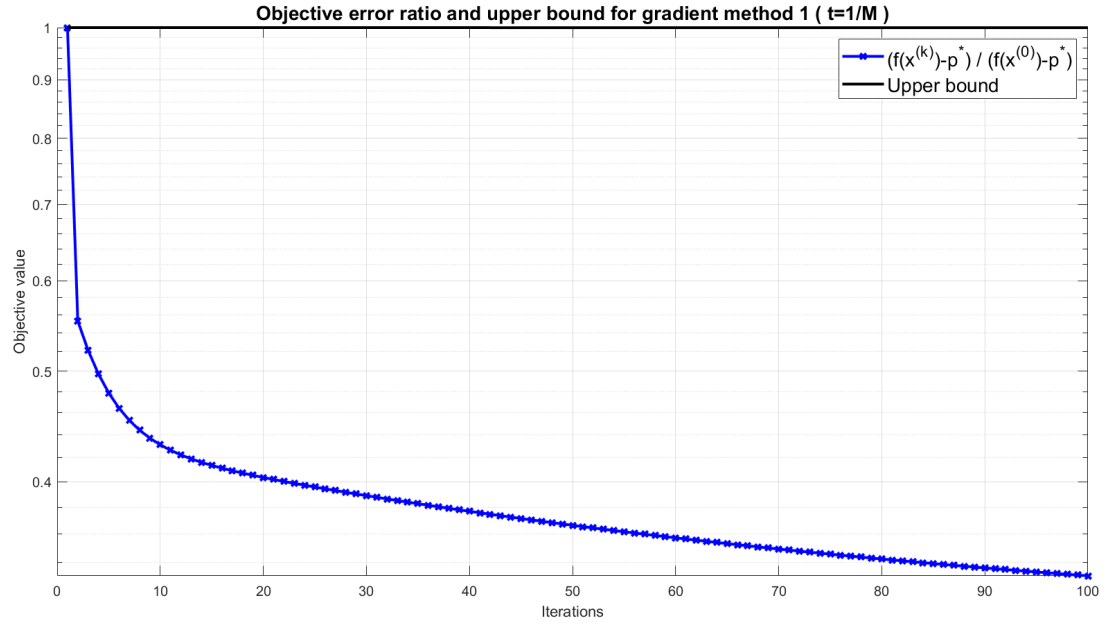


Figure 2: Log plot of the error ratio on the quadratic objective (1) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

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

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

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.

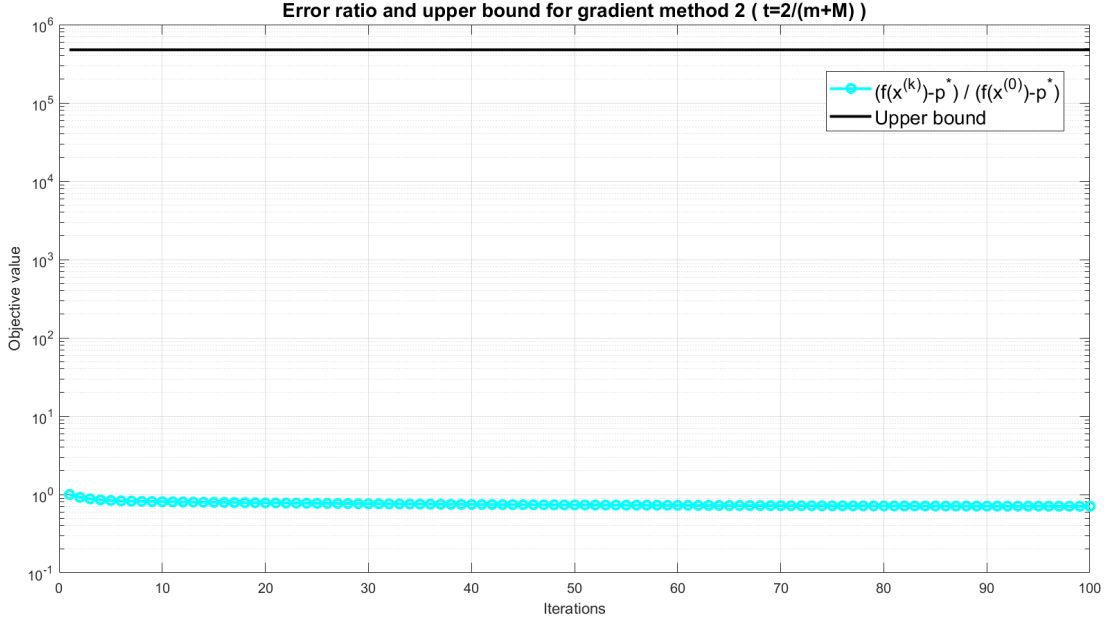
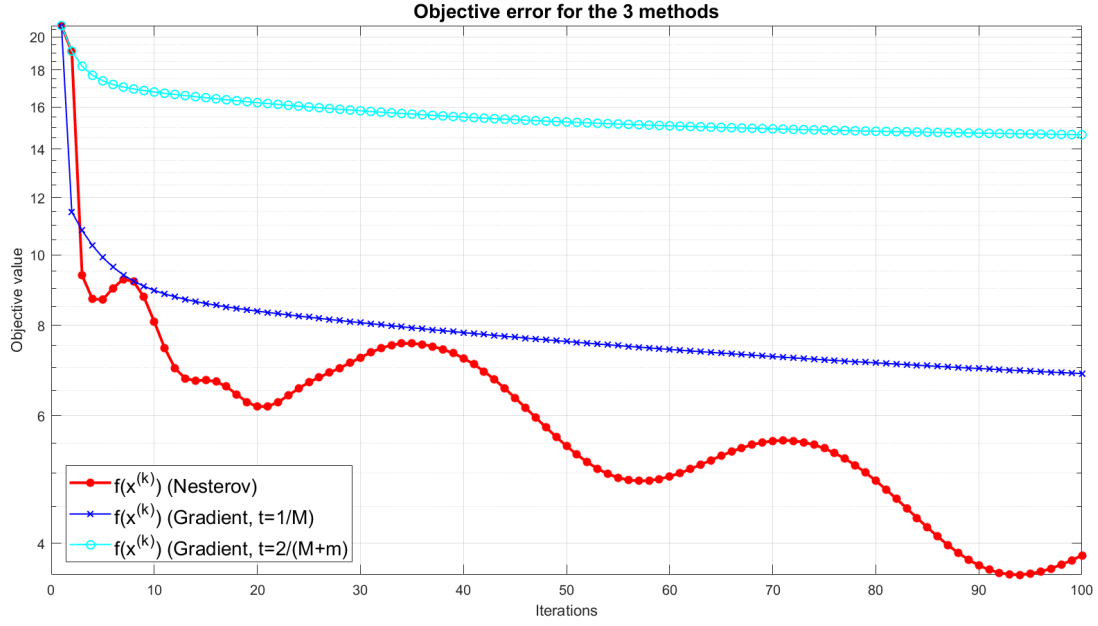


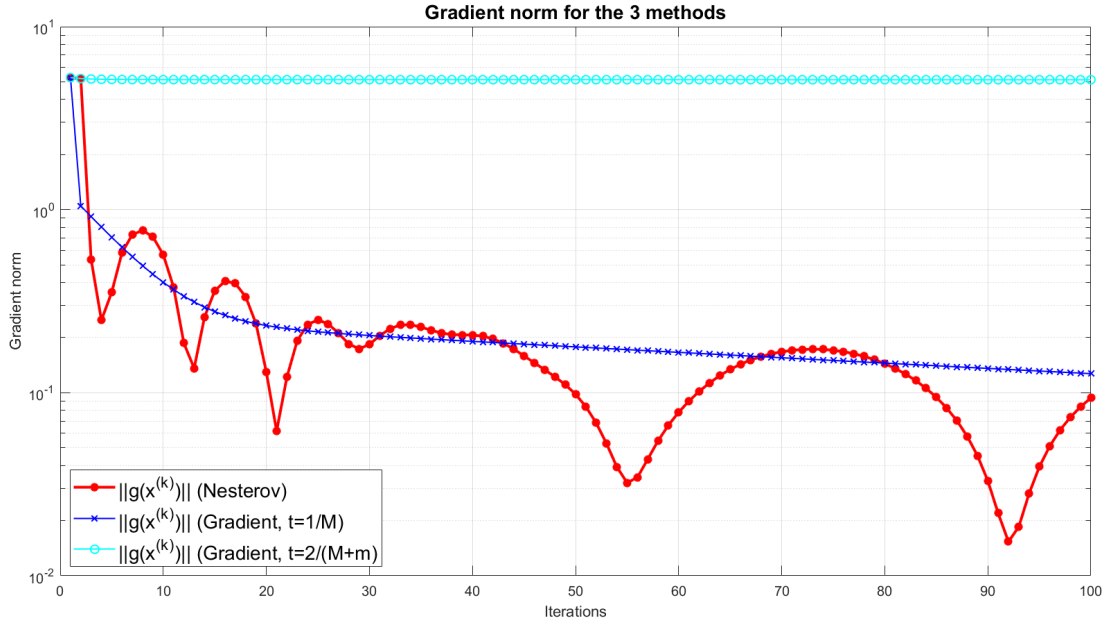
Figure 3: Log plot of the error ratio on the quadratic objective (1) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound

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

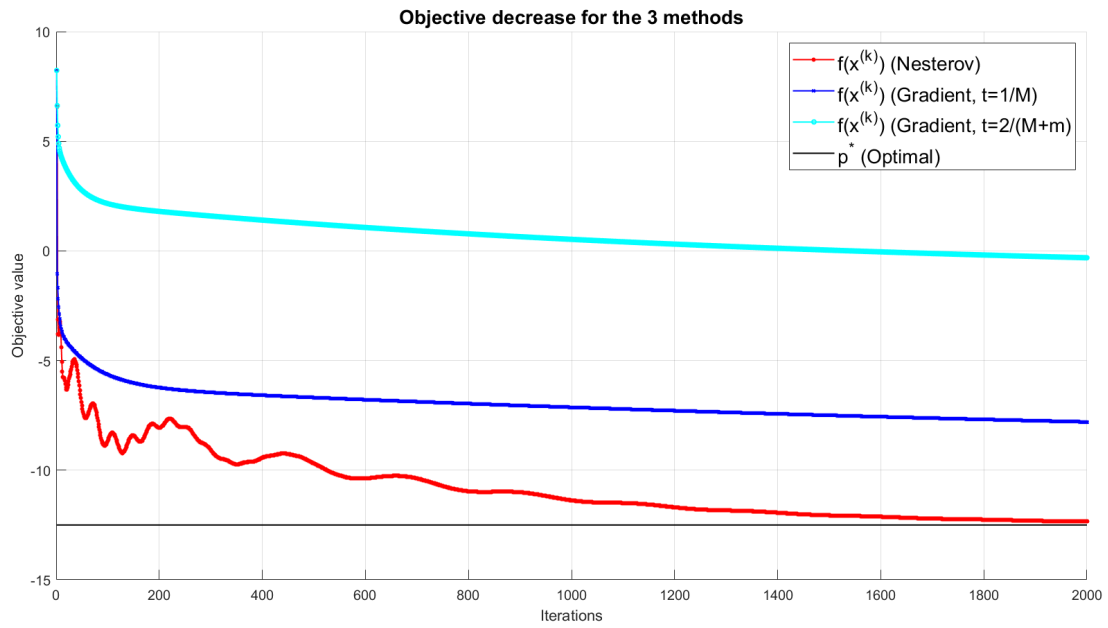


Figure 5: Errors on the quadratic objective (1) minimized by the 3 methods for 2000 iterations.

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 $x^{(0)}, x^{(1)}, \dots$ 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.

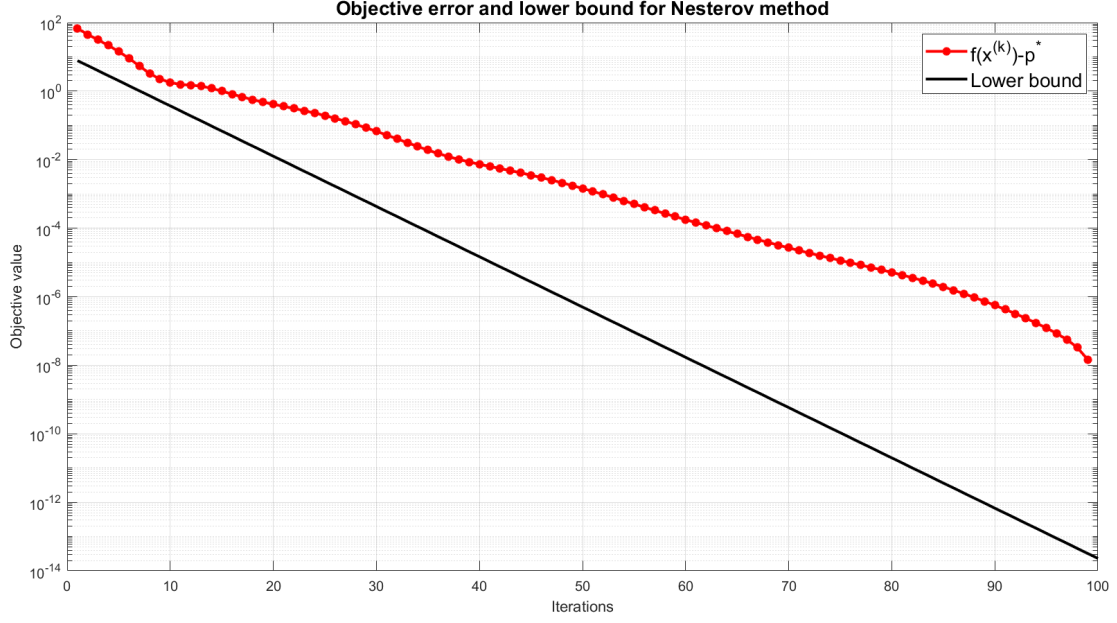


Figure 6: Log plot of the error for the objective BV 9.30 minimized using Nesterov's optimal gradient method and its lower bound

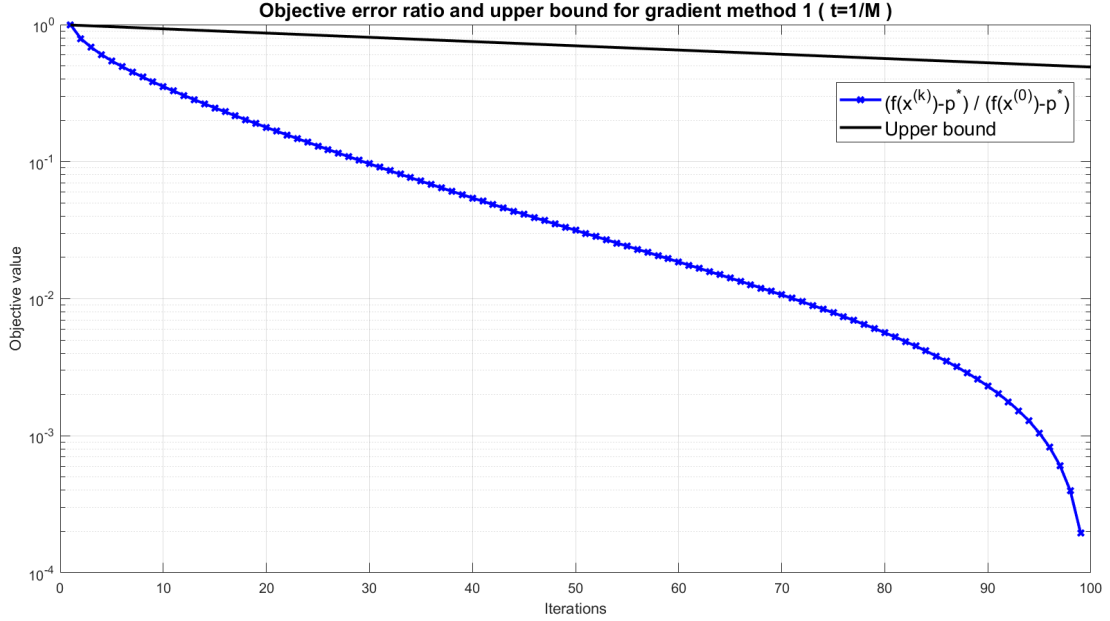


Figure 7: Log plot of the error for the objective BV 9.30 minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

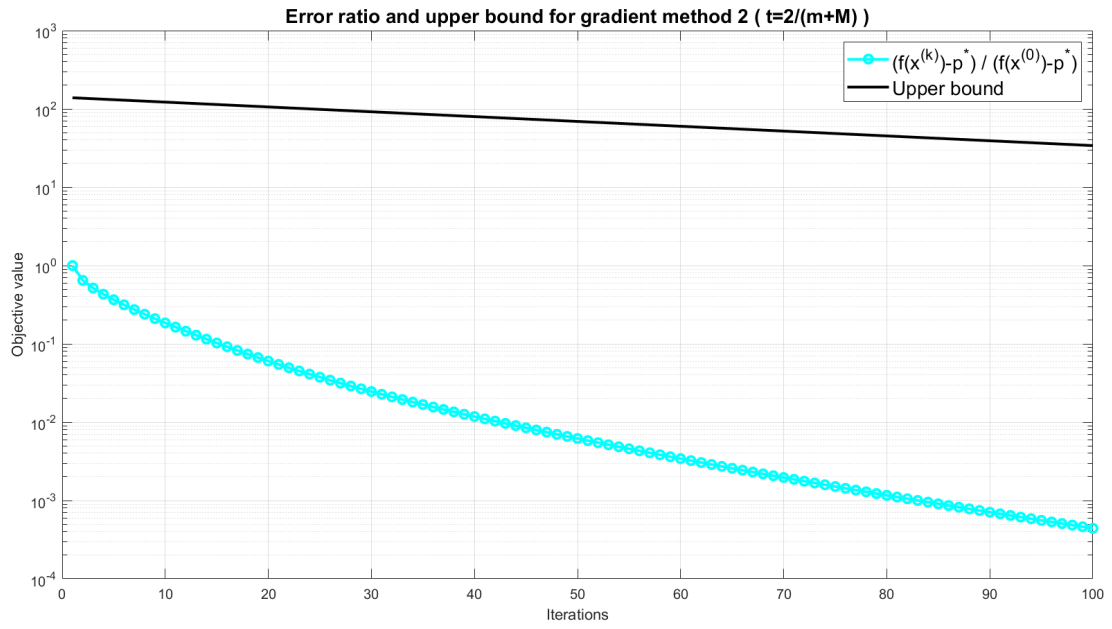
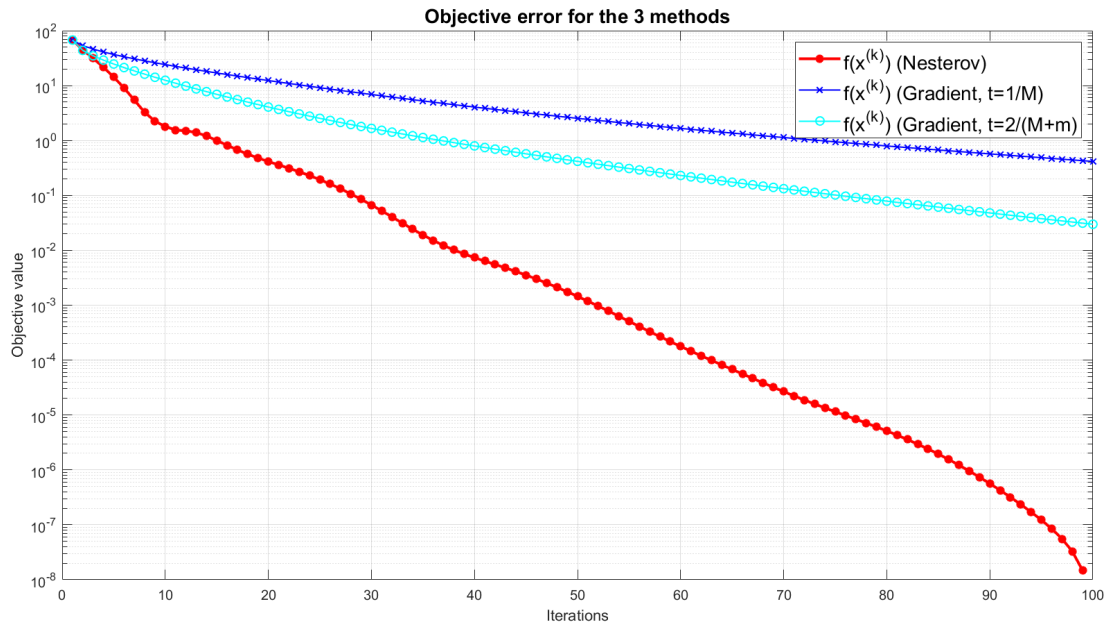
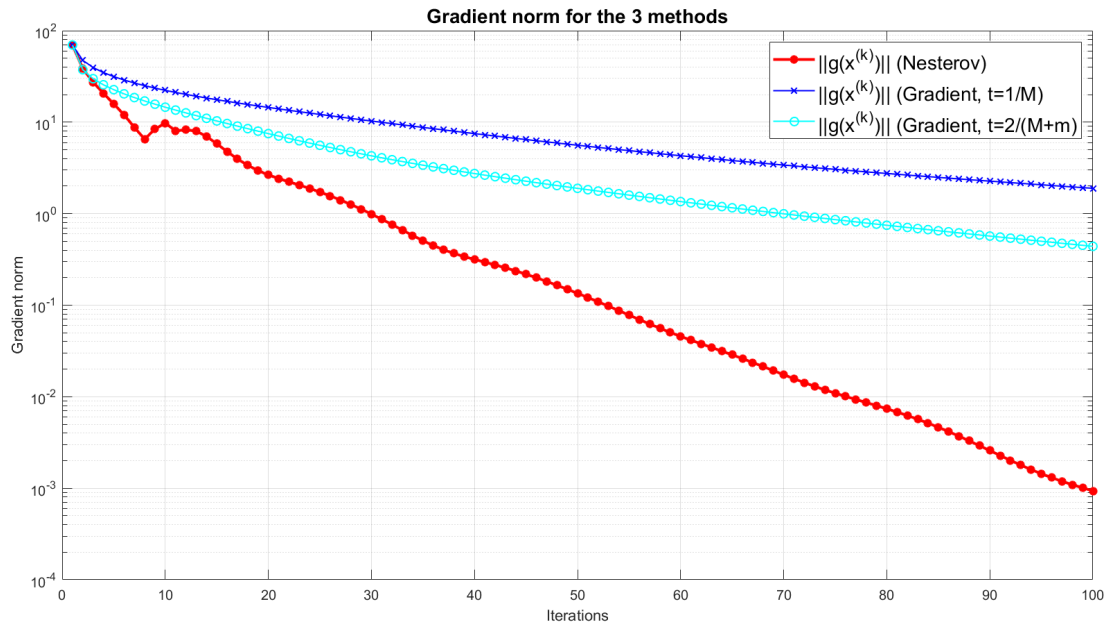


Figure 8: Log plot of the error for the objective BV 9.30 minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound



(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 $j = 3 \dots n$, $x_{j+1} = q^{j+1}$. Then for x_1 and x_2 , knowing $x_3 = q^3$, we solve

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

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.

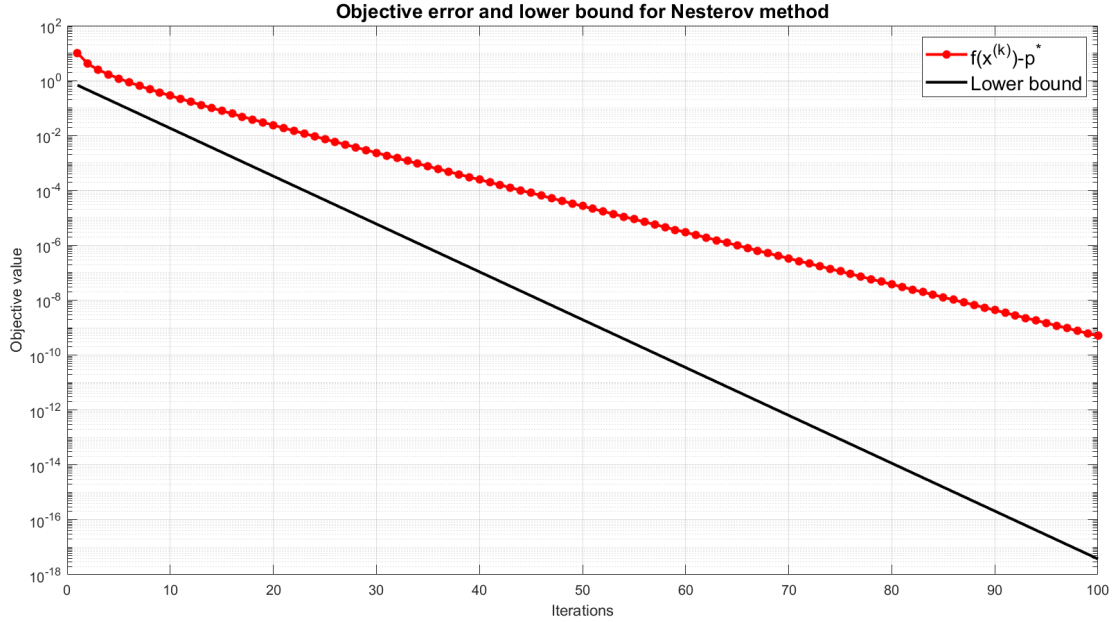


Figure 10: Log plot of the error for Nesterov's example ($M = 100$) minimized using Nesterov's optimal gradient method and its lower bound

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.

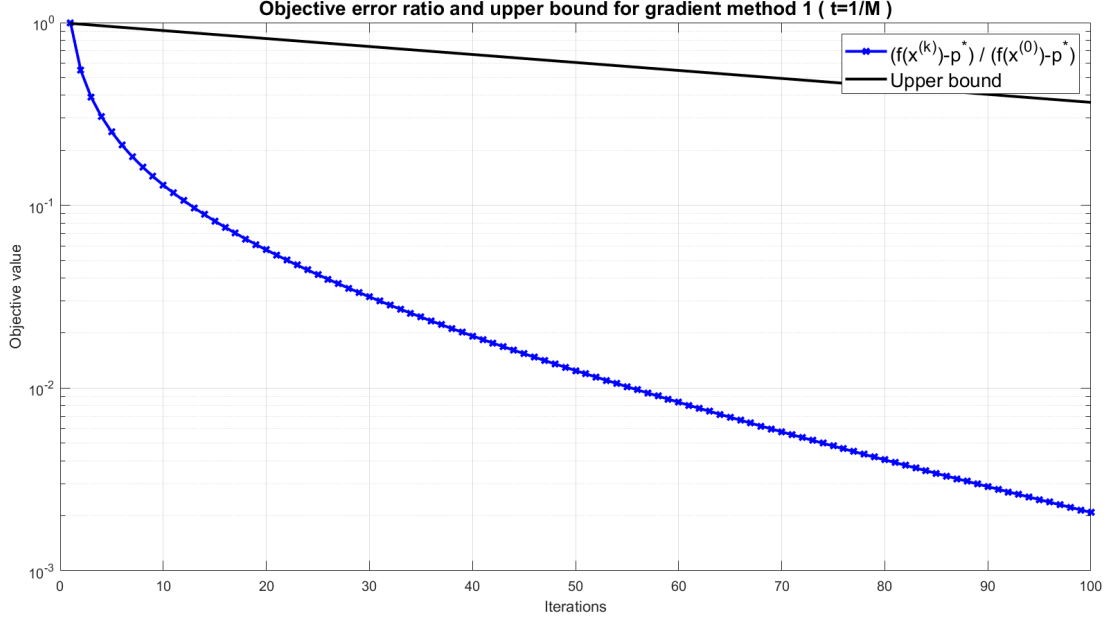


Figure 11: Log plot of the error for Nesterov's example ($M = 100$) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

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 than the fixed-step gradient descent with $t = \frac{2}{M+m}$, which itself outperforms $t = \frac{1}{M}$.

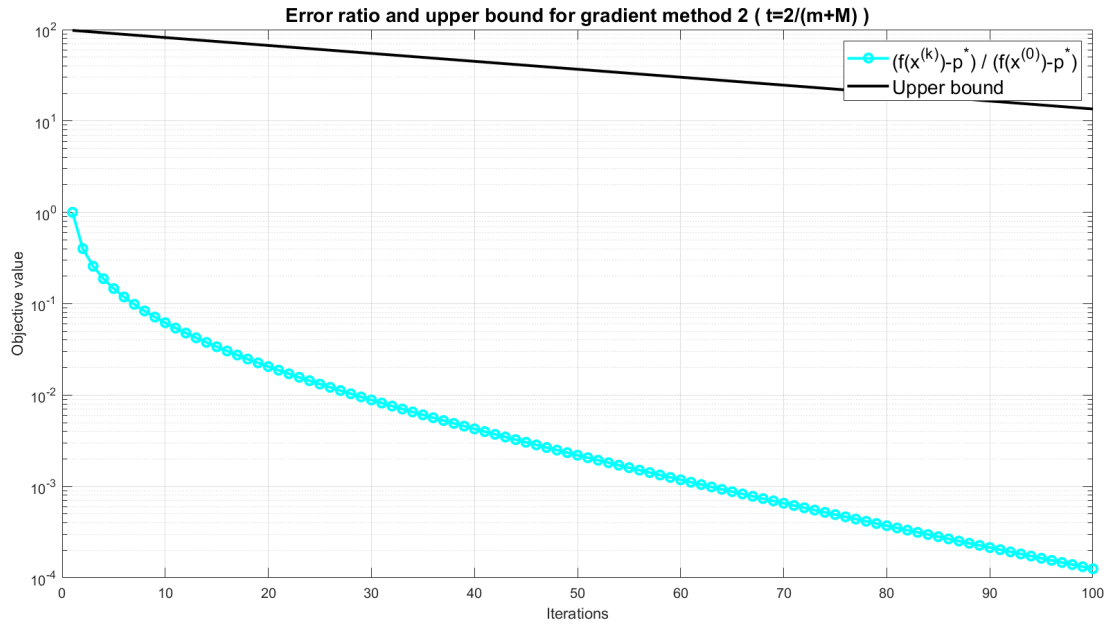
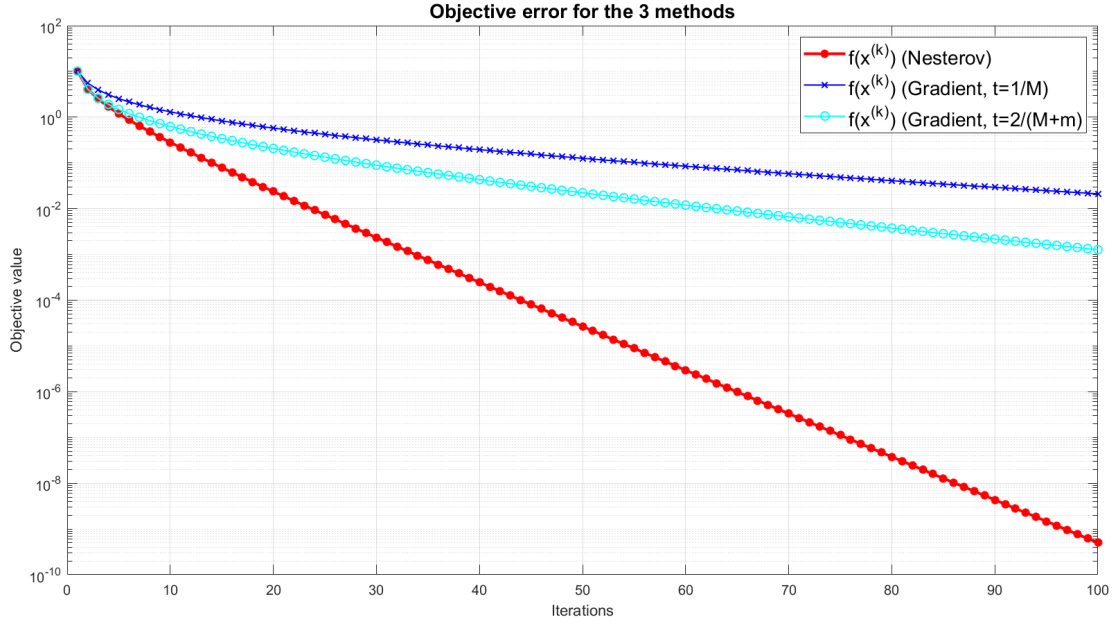
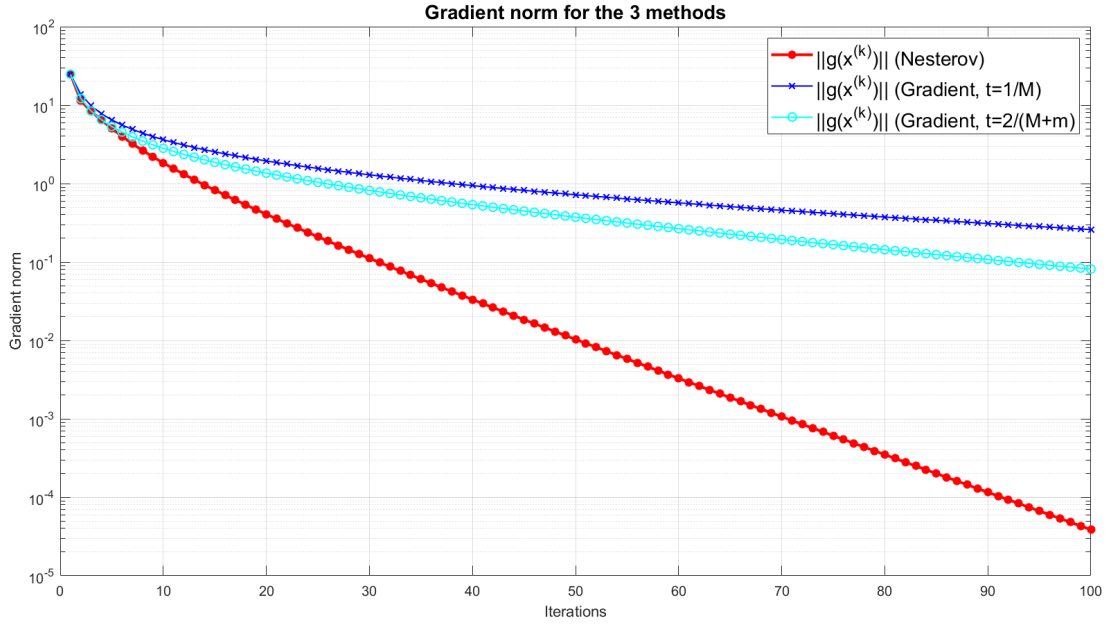


Figure 12: Log plot of the error for Nesterov's example ($M = 100$) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound



(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.

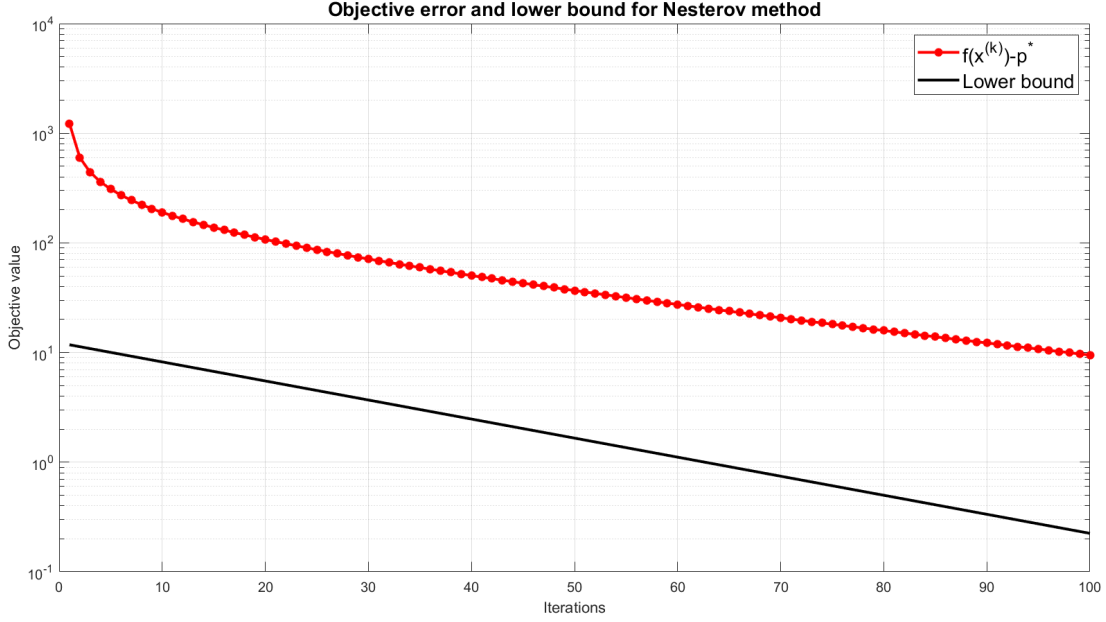


Figure 14: Log plot of the error for Nesterov's example ($M = 10000$) minimized using Nesterov's optimal gradient method and its lower bound

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

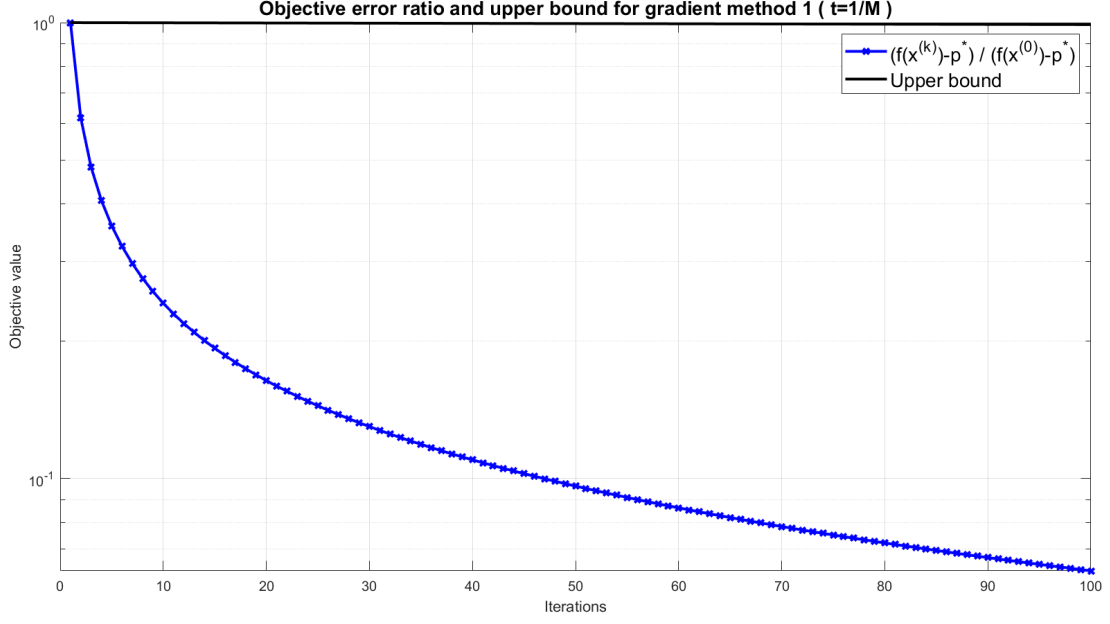


Figure 15: Log plot of the error for Nesterov's example ($M = 10000$) minimized using gradient descent with $t = \frac{1}{M}$ and upper bound

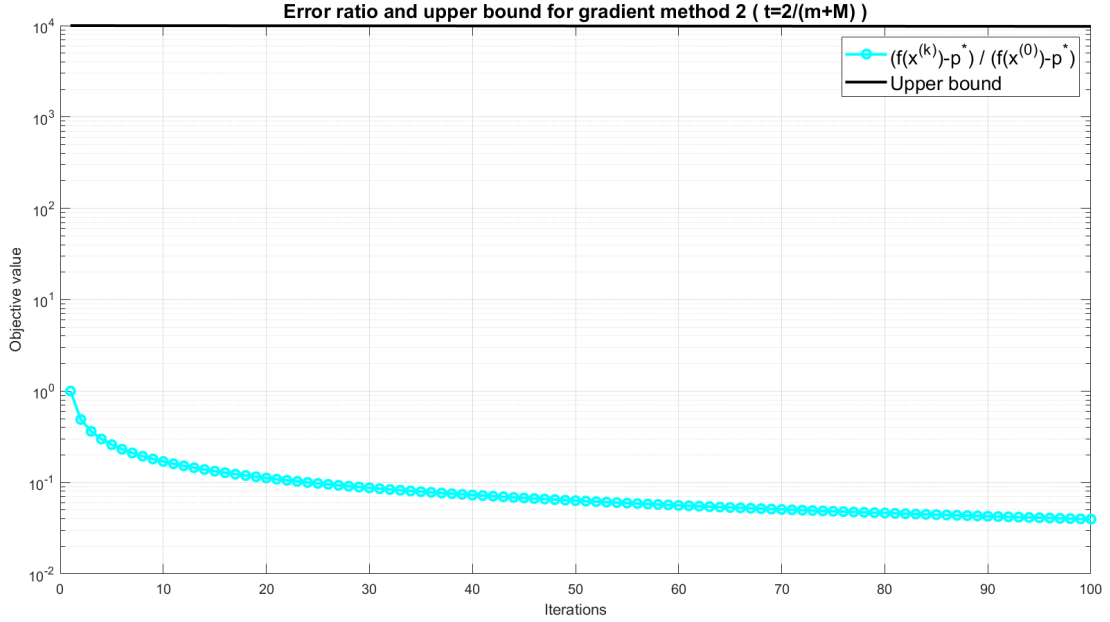
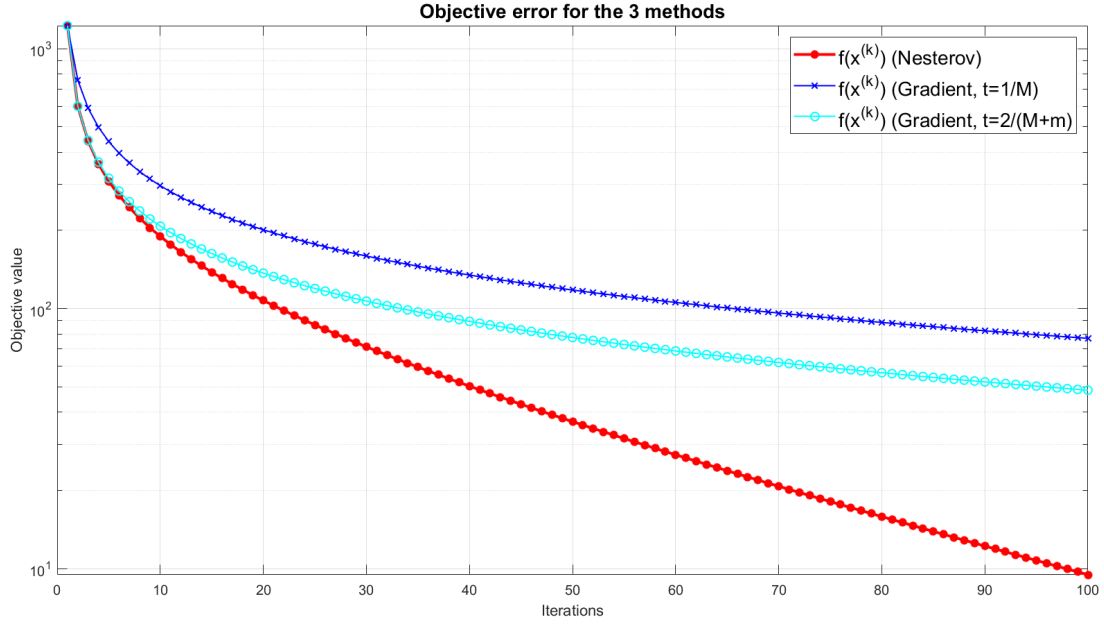
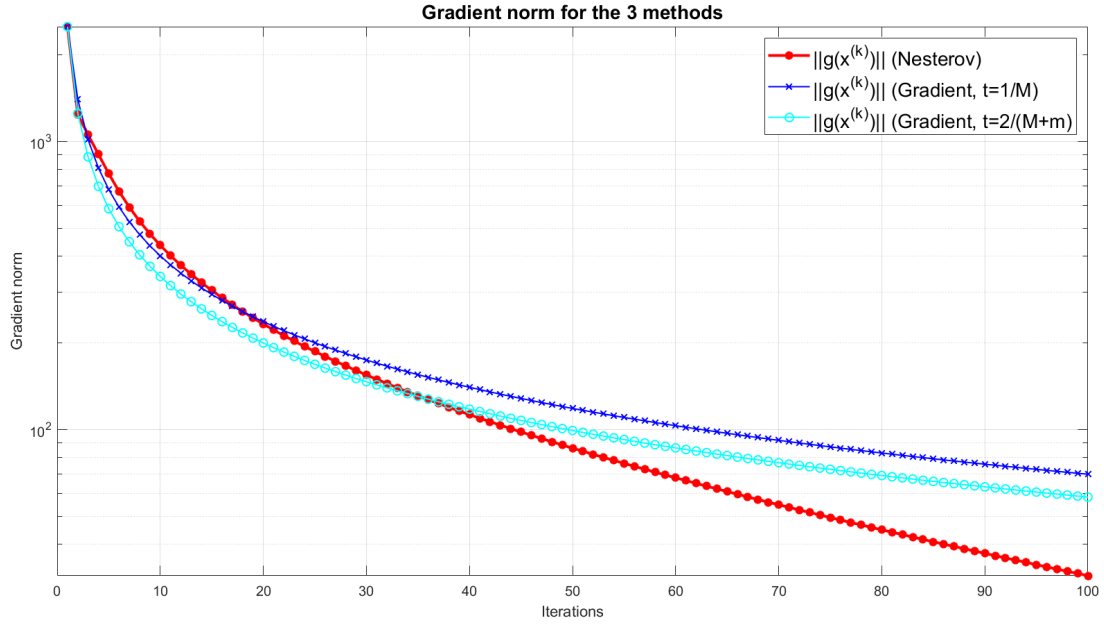


Figure 16: Log plot of the error for Nesterov's example ($M = 10000$) minimized using gradient descent with $t = \frac{2}{m+M}$ and upper bound

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)

```
1 function [f, g, h, x] = hilbert_fun(x,A,b)
2 % quadratic function f(x) = 1/2 x'Ax + b'x
3 % returns value, gradient and hessian and x
4
5 % INPUT : 'x' : current point
6 %         'A' : quadratic term
7 %         'b' : linear term
8
9 % OUTPUT : 'f' : objective function value at x
10 %          'g' : gradient at x
11 %          'h' : hessian at x
12 %          'x' : x
13
14 f = 0.5*x'*A*x + b'*x;
15 g = A*x + b;
16 h = A;
```

```
1 %% HW 6 – Quadratic problem of HW5 (Hilbert matrix)
2
3 % We define an ill-conditioned matrix A and some vector b
4 n = 5;
5 A = hilb(n);
6 b = ones(n,1);
7
8 % The quadratic objective function is defined in quad.m
9 fun = @(x)hilbert_fun(x, A, b);
10
11 % Closed-form solution to the quadratic optimization problem
12 xstar = A\b;
13 pstar = fun(xstar);
14
15 % Condition number of A
16 M = max(eig(A));
17 m = min(eig(A));
18 kappa = M/m;
19
20 % Starting point & algo parameters (tolerance, #iterations)
21 x0 = ones(n,1);
22 tol = 1e-6;
23 maxit = 2000; %100;
24
25 % Iteration counter for plots (x-axis)
26 t = linspace(1, maxit, maxit);
27
28 %% Nesterov's Optimal Gradient (accelerated gradient)
29 % Parameter q
30 q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );
31
32 % Launch Nesterov algorithm
33 [f_all_nest, gnorm_all_nest] = nesterov(fun, x0, M, q, tol, maxit);
34
35 % Verify the lower bound of Nesterov
```

```

36 % Plot objective error and lower bound in a LOG plot
37 figure
38     % Plot error
39 err_nest = f_all_nest - pstar;
40 semilogy(err_nest, 'r-*','LineWidth',1,'MarkerSize',3);
41 hold on
42     % Plot lower bound
43 lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
44 semilogy(lower_bd_nest, 'k-', 'LineWidth', 1);
45     % Custom
46 grid on
47 title('Objective error and lower bound for Nesterov method',...
48     'FontSize', 14);
49 xlabel('Iterations');
50 ylabel('Objective value');
51 legend('f(x^{(k)})-p^{*}', ...
52     'Lower bound', ...
53     'FontSize', 14, 'Location', 'NorthEast');
54
55 %% Gradient descent with fixed step size t = 1/M
56 % Step size
57 t1 = 1/M;
58
59 % Launch the gradient descent implemented in gradmeth.m
60 [f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);
61
62 % Verify convergence
63 % Plot objective error ratio and upper bound in a LOG plot
64 figure
65     % Plot error ratio
66 err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
67 semilogy(err_ratio_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
68 hold on
69     % Plot upper bound
70 upper_bd_grad1 = (1-1/kappa).^t;
71 semilogy(upper_bd_grad1, 'k-', 'LineWidth', 1);
72     % Custom
73 grid on
74 title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
75     'FontSize', 14);
76 xlabel('Iterations');
77 ylabel('Objective value');
78 legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
79     'Upper bound', ...
80     'FontSize', 14, 'Location', 'NorthEast');
81
82 %% Gradient descent with fixed step size = t = 2/(M+m)
83 % Step size
84 t2 = 2/(M+m);
85
86 % Launch the gradient descent implemented in gradmeth.m
87 [f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);
88
89 % Verify convergence properties
90 % Plot objective error ratio and upper bound in a LOG plot
91 figure
92     % Plot error ratio
93 err_ratio_grad2 = ( f_all_grad2 - pstar ) / (f_all_grad2(1) - pstar);

```

```

94 semilogy(err_ratio_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
95 hold on
96     % Plot upper bound
97 upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^(2*t);
98 semilogy(upper_bd_grad2, 'k-', 'LineWidth', 1);
99     % Custom
100 grid on
101 title('Error ratio and upper bound for gradient method 2 ( t=2/(m+M) )',...
102     'FontSize', 14);
103 xlabel('Iterations');
104 ylabel('Objective value');
105 legend(' (f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*}) ', ...
106     'Upper bound', ...
107     'FontSize', 14, 'Location', 'NorthEast');
108
109
110 %% Plot objective for 3 methods in same plot
111 % Objective
112 figure, hold on, grid on
113 % Nesterov
114 plot(t, f_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
115 % Gradient 1
116 plot(t, f_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
117 % Gradient 2
118 plot(t, f_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
119 % Optimal
120 plot(t, pstar*ones(1,maxit),'k-', 'LineWidth',1);
121 % Legend, axis and title
122 title('Objective decrease for the 3 methods',...
123     'FontSize', 14);
124 xlabel('Iterations');
125 ylabel('Objective value');
126 legend('f(x^{(k)}) (Nesterov)', ...
127     'f(x^{(k)}) (Gradient, t=1/M)', ...
128     'f(x^{(k)}) (Gradient, t=2/(M+m))', ...
129     'p^{*} (Optimal)', ...
130     'FontSize', 14, 'Location', 'NorthEast');
131
132 %% Plot gradient norm for 3 methods in same plot
133 % Gradient norm
134 figure, hold on, grid on
135 % Nesterov
136 plot(t, gnorm_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
137 % Gradient 1
138 plot(t, gnorm_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
139 % Gradient 2
140 plot(t, gnorm_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
141 % Legend, axis and title
142 title('Gradient norms for the 3 methods',...
143     'FontSize', 14);
144 xlabel('Iterations');
145 ylabel('Gradient norm');
146 legend('||g(x^{(k)})|| (Nesterov)', ...
147     '||g(x^{(k)})|| (Gradient, t=1/M)', ...
148     '||g(x^{(k)})|| (Gradient, t=2/(M+m))', ...
149     'FontSize', 14, 'Location', 'NorthEast');
150
151 %% Plot objective error for 3 methods in same LOG plot

```

```

152 % Objective log error
153 figure
154 % Nesterov
155 semilogy(t, f_all_nest - pstar, 'r-*','LineWidth',1,'MarkerSize',3);
156 hold on
157 % Gradient 1
158 semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1,'MarkerSize',3);
159 % Gradient 2
160 semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1,'MarkerSize',3);
161 % Legend, axis and title
162 title('Objective error for the 3 methods',...
163       'FontSize', 14);
164 xlabel('Iterations');
165 ylabel('Objective value');
166 legend('f(x^{k}) (Nesterov)', ...
167        'f(x^{k}) (Gradient, t=1/M)', ...
168        'f(x^{k}) (Gradient, t=2/(M+m))', ...
169        'FontSize', 14, 'Location', 'NorthEast');
170 grid on
171
172 %% Plot gradient norm for 3 methods in same LOG plot
173 % Gradient norm log plot
174 figure
175 % Nesterov
176 semilogy(t, gnorm_all_nest, 'r-*','LineWidth',1,'MarkerSize',3);
177 hold on
178 % Gradient 1
179 semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1,'MarkerSize',3);
180 % Gradient 2
181 semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1,'MarkerSize',3);
182 % Legend, axis and title
183 title('Gradient norm for the 3 methods',...
184       'FontSize', 14);
185 xlabel('Iterations');
186 ylabel('Gradient norm');
187 legend('||g(x^{k})|| (Nesterov)', ...
188        '||g(x^{k})|| (Gradient, t=1/M)', ...
189        '||g(x^{k})|| (Gradient, t=2/(M+m))', ...
190        'FontSize', 14, 'Location', 'NorthEast');
191 grid on

```

B MATLAB code for BV 9.30 (Section 2)

```

1 function [f, g, h, x] = bv930_fun(x, a)
2 % Objective function of Ex 9.30 (BV p.519)
3 % This function returns the evaluation, gradient and hessian and x
4
5 % INPUT : 'x' : current point
6 %         'a' : matrix a
7
8 % OUPUT : 'f' : objective function value at x
9 %         'g' : gradient at x
10 %         'h' : hessian at x
11 %         'x' : x

```

```

12
13
14
15 % Sizes
16 n = size(a,1);
17 m = size(a,2);
18
19 % Compute ai'*x, i=1..m
20 ax = zeros(m,1);
21 for i=1:m
22     ax(i) = a(:,i)';
23 end
24
25 % If x is not in dom(f) return default values for f and g
26 if max(ax) ≥ 1 || max(abs(x)) ≥ 1
27     f = inf;
28     g = nan*ones(n,1);
29
30 % Else continue
31 else
32 %% Calculate f := f(x)
33     f=0;
34     % First term of f : sum over m terms with ai'*x
35     for i=1:m
36         f = f - log(1-ax(i));
37     end
38     % Second term of f : sum over n terms with xi^2
39     for i=1:n
40         f = f - log(1-x(i)^2);
41     end
42
43 %% Calculate gradient
44     g = zeros(n,1);
45
46     % Analytic formula for the gradient
47     for i=1:n
48         g(i) = 2*x(i)/(1-x(i)^2);
49         for j=1:m
50             g(i) = g(i) + a(i,j)/(1-a(:,j)');
51         end
52     end
53
54 %% Calculate hessian
55     h = zeros(n,n);
56
57     % Analytic formula for the hessian
58     for i=1:n
59         for k=1:n
60             if i==k
61                 h(i,k) = 2*(1 + x(k)^2) / ( (1 - x(k)^2)^2 );
62             else
63                 h(i,k) = 0;
64             end
65             for j=1:m
66                 h(i,k) = h(i,k) + a(i,j)*a(k,j)/( (1-a(:,j)')^2 );
67             end
68         end
69     end

```

```

70
71 end
72
73 end

```

```

1 %% HW6 – BV 9.30 (p.519)
2
3 % Load data set for BV 9.30
4 A = load('Adata.mat').A;
5 n = size(A,1);
6
7 % The objective function of BV 9.30 is implemented in bv930_fun.m
8 fun = @(x)bv930_fun(x, A);
9
10 % Starting point & algo parameters (tolerance, #iterations)
11 x0 = zeros(n,1);
12 tol = 1e-6;
13 maxit = 100;
14
15 % Iteration counter for plots (x-axis)
16 t = linspace(1, maxit, maxit);
17
18 %% Newton's method (ref for pstar)
19 % % Launch Newton's method to evaluate pstar and bounds on Hessian m,M
20 % [f_all, gnorm_all, h_all, x_all] = newtmeth(fun, x0, tol, maxit);
21 % min_eigs = [];
22 % max_eigs = [];
23 % for i=1:length(x_all)
24 %     h = hessian(A, x{i});
25 %     min_eigs = [min_eigs, min(eig(h_all{i}))];
26 %     max_eigs = [max_eigs, max(eig(h_all{i}))];
27 % end
28 % m = min(min_eigs)
29 % M = max(max_eigs)
30 % kappa = M/m
31
32 %% Gradient descent to evaluate (m,M)
33 % Get a 1st guess for m,M with Hessian at starting point
34 [f, g, h, x] = fun(x0);
35 m = min(eig(h));
36 M = max(eig(h));
37 % Launch grad descent with t=1/M to re-estimate bounds from sequence
38 [f_all, gnorm_all, x_all] = gradmeth(fun, x0, 1/M, tol, maxit);
39 min_eigs = [];
40 max_eigs = [];
41 for i=1:length(x_all)
42     h = hessian(A, x_all{i});
43     min_eigs = [min_eigs, min(eig(h))];
44     max_eigs = [max_eigs, max(eig(h))];
45 end
46 m = min(min_eigs);
47 M = max(max_eigs);
48 kappa = M/m;
49
50 % Define optimal value to be last element of Newton sequence
51 pstar = f_all(end);

```

```

52 xstar = x_all{end};
53
54 %% Nesterov's Optimal Gradient (accelerated gradient)
55 % Parameter q
56 q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );
57
58 % Launch Nesterov algorithm
59 [f_all_nest, gnorm_all_nest] = nesterov(fun, x0, M, q, tol, maxit);
60
61 % Estimate optimal value by last element of Nesterov sequence
62 pstar = f_all_nest(end);
63
64 % Verify the lower bound of Nesterov
65 % Plot objective error and lower bound in a LOG plot
66 figure
67     % Plot error
68 err_nest = f_all_nest - pstar;
69 semilogy(err_nest, 'r-*', 'LineWidth', 2);
70 hold on
71     % Plot lower bound
72 lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
73 semilogy(lower_bd_nest, 'k-', 'LineWidth', 2);
74     % Custom
75 grid on
76 title('Objective error and lower bound for Nesterov method',...
77     'FontSize', 14);
78 xlabel('Iterations');
79 ylabel('Objective value');
80 legend('f(x^{(k)})-p^{*}', ...
81     'Lower bound', ...
82     'FontSize', 14, 'Location', 'NorthEast');
83
84 %% Gradient descent with fixed step size t = 1/M
85 % Step size
86 t1 = 1/M;
87
88 % Launch the gradient descent implemented in gradmeth.m
89 [f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);
90
91 % Verify convergence
92 % Plot objective error ratio and upper bound in a LOG plot
93 figure
94     % Plot error ratio
95 err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
96 semilogy(err_ratio_grad1, 'b-x', 'LineWidth', 2);
97 hold on
98     % Plot upper bound
99 upper_bd_grad1 = (1-1/kappa).^t;
100 semilogy(upper_bd_grad1, 'k-', 'LineWidth', 2);
101     % Custom
102 grid on
103 title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
104     'FontSize', 14);
105 xlabel('Iterations');
106 ylabel('Objective value');
107 legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
108     'Upper bound', ...
109     'FontSize', 14, 'Location', 'NorthEast');

```



```

110
111 %% Gradient descent with fixed step size =  $t = 2/(M+m)$ 
112 % Step size
113  $t2 = 2/(M+m);$ 
114
115 % Launch the gradient descent implemented in gradmeth.m
116 [f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);
117
118 % Verify convergence properties
119 % Plot objective error ratio and upper bound in a LOG plot
120 figure
121     % Plot error ratio
122 err_ratio_grad2 = ( f_all_grad2 - pstar ) / ( f_all_grad2(1) - pstar);
123 semilogy(err_ratio_grad2, 'c-o', 'LineWidth', 2);
124 hold on
125     % Plot upper bound
126 upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^t;
127 semilogy(upper_bd_grad2, 'k-', 'LineWidth', 2);
128     % Custom
129 grid on
130 title('Error ratio and upper bound for gradient method 2 (  $t=2/(m+M)$  )',...
131     'FontSize', 14);
132 xlabel('Iterations');
133 ylabel('Objective value');
134 legend('f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
135     'Upper bound', ...
136     'FontSize', 14, 'Location', 'NorthEast');
137
138
139 %% Plot objective for 3 methods in same plot
140 % Objective
141 figure, hold on, grid on
142 % Nesterov
143 plot(t, f_all_nest, 'r-*, 'LineWidth', 2);
144 % Gradient 1
145 plot(t, f_all_grad1, 'b-x', 'LineWidth', 1);
146 % Gradient 2
147 plot(t, f_all_grad2, 'c-o', 'LineWidth', 1);
148 % Optimal
149 plot(t, pstar*ones(1,maxit), 'k-', 'LineWidth', 2);
150 % Legend, axis and title
151 title('Objective decrease for the 3 methods',...
152     'FontSize', 14);
153 xlabel('Iterations');
154 ylabel('Objective value');
155 legend('f(x^{(k)}) (Nesterov)', ...
156     'f(x^{(k)}) (Gradient,  $t=1/M$ )', ...
157     'f(x^{(k)}) (Gradient,  $t=2/(M+m)$ )', ...
158     'p^{*} (Optimal)', ...
159     'FontSize', 14, 'Location', 'NorthEast');
160
161 %% Plot gradient norm for 3 methods in same plot
162 % Gradient norm
163 figure, hold on, grid on
164 % Nesterov
165 plot(t, gnorm_all_nest, 'r-*, 'LineWidth', 2);
166 % Gradient 1
167 plot(t, gnorm_all_grad1, 'b-x', 'LineWidth', 1);

```

```

168 % Gradient 2
169 plot(t, gnorm_all_grad2, 'c-o','LineWidth',1);
170 % Legend, axis and title
171 title('Gradient norms for the 3 methods',...
172       'FontSize', 14);
173 xlabel('Iterations');
174 ylabel('Gradient norm');
175 legend('||g(x^{k})|| (Nesterov)', ...
176        '||g(x^{k})|| (Gradient, t=1/M)', ...
177        '||g(x^{k})|| (Gradient, t=2/(M+m))', ...
178        'FontSize', 14, 'Location', 'NorthEast');
179
180 %% Plot objective error for 3 methods in same LOG plot
181 % Objective log error
182 figure
183 % Nesterov
184 semilogy(t, f_all_nest - pstar, 'r-','LineWidth',2);
185 hold on
186 % Gradient 1
187 semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1);
188 % Gradient 2
189 semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1);
190 % Legend, axis and title
191 title('Objective error for the 3 methods',...
192       'FontSize', 14);
193 xlabel('Iterations');
194 ylabel('Objective value');
195 legend('f(x^{k}) (Nesterov)', ...
196        'f(x^{k}) (Gradient, t=1/M)', ...
197        'f(x^{k}) (Gradient, t=2/(M+m))', ...
198        'FontSize', 14, 'Location', 'NorthEast');
199 grid on
200
201 %% Plot gradient norm for 3 methods in same LOG plot
202 % Gradient norm log plot
203 figure
204 % Nesterov
205 semilogy(t, gnorm_all_nest, 'r-','LineWidth',2);
206 hold on
207 % Gradient 1
208 semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1);
209 % Gradient 2
210 semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1);
211 % Legend, axis and title
212 title('Gradient norm for the 3 methods',...
213       'FontSize', 14);
214 xlabel('Iterations');
215 ylabel('Gradient norm');
216 legend('||g(x^{k})|| (Nesterov)', ...
217        '||g(x^{k})|| (Gradient, t=1/M)', ...
218        '||g(x^{k})|| (Gradient, t=2/(M+m))', ...
219        'FontSize', 14, 'Location', 'NorthEast');
220 grid on

```

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

```

1 function [f, g, x] = nestex.fun(x, m, M)
2 % Objective function of Nesterov's worst case example (p.67)
3 % This function returns the evaluation, gradient and hessian and x
4
5 % Sizes
6 n = size(x,1);
7
8 % Value of objective
9 f = ( (M-m)/8 ) * ( x(1)^2 - 2*x(1) ) + (m/2)*norm(x,2)^2;
10 for i=1:n-1
11     f = f + ( (M-m)/8 ) * ( x(i)-x(i+1) )^2;
12 end
13
14 % Sparse tridiagonal matrix
15 % T = sparse(full(gallery('tridiag',n,-1,2,-1)));
16
17 % e1
18 e1 = zeros(n,1);
19 e1(1) = 1;
20
21 % Gradient
22 g = sparse( ( (M-m)/4)*T + m*eye(n) )*x - ( (M-m)/4)*e1 );
23
24 T = zeros(n, n);
25 T(1:1+n:n*n) = 2;
26 T(n+1:1+n:n*n) = -1;
27 T(2:1+n:n*n) = -1;
28 T = sparse(T);
29
30 e = zeros(n,1);
31 e(1) = 1;
32 g = sparse( ( (M-m)/4)*T + m*eye(n) )*x - ( (M-m)/4)*e );
33 end

```

```

1 %% HW6 - Nesterov's example (p.67) - M=100 & M=10000
2
3 % Init size and bounds
4 n = 10000;
5 m = 1;
6 M = 100; % or M=10000;
7 kappa = M/m;
8
9 % The objective function of BV 9.30 is implemented in bv930_fun.m
10 fun = @(x)nestex.fun(x, m, M);
11
12 % Starting point & algo parameters (tolerance, #iterations)
13 x0 = zeros(n,1);
14 size(x0)
15 tol = 1e-6;
16 maxit = 100;
17
18 % Iteration counter for plots (x-axis)
19 t = linspace(1, maxit, maxit);
20
21 %% Nesterov's Optimal Gradient (accelerated gradient)
22 % Parameter q

```

```

23 q = ( 1 - sqrt(1/kappa) ) / ( 1 + sqrt(1/kappa) );
24
25 % Calculate optimal value (analytical solution taken from lecture notes)
26 % I solved the linear system for x(1) and x(2) using "\" in symbolic
27 xtsar = zeros(n,1);
28 % For the rest of the vector, it's simply x(i) = q^i
29 for i=3:n
30     xstar(i,1) = q^i;
31 end
32 tmp = (M + m) / (M-m);
33 xstar(2,1) = (2*tmp*xstar(3)+1)/(4*tmp^2-1);
34 xstar(1,1) = (1 + xstar(2))/(2*tmp);
35 [pstar,gstar,xstar] = fun(xstar);
36
37 % Launch Nesterov algorithm
38 [f_all_nest, gnorm_all_nest, x_all_nest] = nesterov(fun, x0, M, q, tol, maxit);
39
40 % Verify the lower bound of Nesterov
41 % Plot objective error and lower bound in a LOG plot
42 figure
43     % Plot error
44 err_nest = f_all_nest - pstar;
45 semilogy(err_nest, 'r-*', 'LineWidth', 2);
46 hold on
47     % Plot lower bound
48 lower_bd_nest = (m/2)*q.^(2*t)*(norm(x0 - xstar)^2);
49 semilogy(lower_bd_nest, 'k-', 'LineWidth', 2);
50     % Custom
51 grid on
52 title('Objective error and lower bound for Nesterov method',...
53     'FontSize', 14);
54 xlabel('Iterations');
55 ylabel('Objective value');
56 legend('f(x^{(k)})-p^{*}', ...
57     'Lower bound', ...
58     'FontSize', 14, 'Location', 'NorthEast');
59
60 %% Gradient descent with fixed step size t = 1/M
61 % Step size
62 t1 = 1/M;
63
64 % Launch the gradient descent implemented in gradmeth.m
65 [f_all_grad1, gnorm_all_grad1] = gradmeth(fun, x0, t1, tol, maxit);
66
67 % Verify convergence
68 % Plot objective error ratio and upper bound in a LOG plot
69 figure
70     % Plot error ratio
71 err_ratio_grad1 = ( f_all_grad1 - pstar ) / (f_all_grad1(1) - pstar);
72 semilogy(err_ratio_grad1, 'b-x', 'LineWidth', 2);
73 hold on
74     % Plot upper bound
75 upper_bd_grad1 = (1-1/kappa).^t;
76 semilogy(upper_bd_grad1, 'k-', 'LineWidth', 2);
77     % Custom
78 grid on
79 title('Objective error ratio and upper bound for gradient method 1 ( t=1/M )',...
80     'FontSize', 14);

```

```

81 xlabel('Iterations');
82 ylabel('Objective value');
83 legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
84         'Upper bound', ...
85         'FontSize', 14, 'Location', 'NorthEast');
86
87 %% Gradient descent with fixed step size = t = 2/(M+m)
88 % Step size
89 t2 = 2/(M+m);
90
91 % Launch the gradient descent implemented in gradmeth.m
92 [f_all_grad2, gnorm_all_grad2] = gradmeth(fun, x0, t2, tol, maxit);
93
94 % Verify convergence properties
95 % Plot objective error ratio and upper bound in a LOG plot
96 figure
97     % Plot error ratio
98 err_ratio_grad2 = ( f_all_grad2 - pstar ) / ( f_all_grad2(1) - pstar );
99 semilogy(err_ratio_grad2, 'c-o', 'LineWidth', 2);
100 hold on
101     % Plot upper bound
102 upper_bd_grad2 = kappa*( (1-1/kappa) / (1+1/kappa) ).^t;
103 semilogy(upper_bd_grad2, 'k-', 'LineWidth', 2);
104     % Custom
105 grid on
106 title('Error ratio and upper bound for gradient method 2 ( t=2/(m+M) )',...
107        'FontSize', 14);
108 xlabel('Iterations');
109 ylabel('Objective value');
110 legend('(f(x^{(k)})-p^{*}) / (f(x^{(0)})-p^{*})', ...
111        'Upper bound', ...
112        'FontSize', 14, 'Location', 'NorthEast');
113
114
115 %% Plot objective for 3 methods in same plot
116 %% Objective
117 % figure, hold on, grid on
118 %% Nesterov
119 % plot(t, f_all_nest, 'r-*','LineWidth',2);
120 %% Gradient 1
121 % plot(t, f_all_grad1, 'b-x','LineWidth',1);
122 %% Gradient 2
123 % plot(t, f_all_grad2, 'c-o','LineWidth',1);
124 %% Optimal
125 % plot(t, pstar*ones(1,maxit),'k-','LineWidth',2);
126 %% Legend, axis and title
127 % title('Objective decrease for the 3 methods',...
128 %       'FontSize', 14);
129 % xlabel('Iterations');
130 % ylabel('Objective value');
131 % legend('f(x^{(k)}) (Nesterov)', ...
132 %       'f(x^{(k)}) (Gradient, t=1/M)', ...
133 %       'f(x^{(k)}) (Gradient, t=2/(M+m))', ...
134 %       'p^{*} (Optimal)', ...
135 %       'FontSize', 14, 'Location', 'NorthEast');
136
137 %% Plot gradient norm for 3 methods in same plot
138 %% Gradient norm

```

```

139 % figure, hold on, grid on
140 % % Nesterov
141 % plot(t, gnorm_all_nest, 'r-*','LineWidth',2);
142 % % Gradient 1
143 % plot(t, gnorm_all_grad1, 'b-x','LineWidth',1);
144 % % Gradient 2
145 % plot(t, gnorm_all_grad2, 'c-o','LineWidth',1);
146 % % Legend, axis and title
147 % title('Gradient norms for the 3 methods',...
148 %       'FontSize', 14);
149 % xlabel('Iterations');
150 % ylabel('Gradient norm');
151 % legend('||g(x^{k})|| (Nesterov)', ...
152 %       '||g(x^{k})|| (Gradient, t=1/M)', ...
153 %       '||g(x^{k})|| (Gradient, t=2/(M+m))', ...
154 %       'FontSize', 14, 'Location', 'NorthEast');
155
156 %% Plot objective error for 3 methods in same LOG plot
157 % Objective log error
158 figure
159 % Nesterov
160 semilogy(t, f_all_nest - pstar, 'r-*','LineWidth',2);
161 hold on
162 % Gradient 1
163 semilogy(t, f_all_grad1 - pstar, 'b-x','LineWidth',1);
164 % Gradient 2
165 semilogy(t, f_all_grad2 - pstar, 'c-o','LineWidth',1);
166 % Legend, axis and title
167 title('Objective error for the 3 methods',...
168       'FontSize', 14);
169 xlabel('Iterations');
170 ylabel('Objective value');
171 legend('f(x^{k}) (Nesterov)', ...
172       'f(x^{k}) (Gradient, t=1/M)', ...
173       'f(x^{k}) (Gradient, t=2/(M+m))', ...
174       'FontSize', 14, 'Location', 'NorthEast');
175 grid on
176
177 %% Plot gradient norm for 3 methods in same LOG plot
178 % Gradient norm log plot
179 figure
180 % Nesterov
181 semilogy(t, gnorm_all_nest, 'r-*','LineWidth',2);
182 hold on
183 % Gradient 1
184 semilogy(t, gnorm_all_grad1, 'b-x','LineWidth',1);
185 % Gradient 2
186 semilogy(t, gnorm_all_grad2, 'c-o','LineWidth',1);
187 % Legend, axis and title
188 title('Gradient norm for the 3 methods',...
189       'FontSize', 14);
190 xlabel('Iterations');
191 ylabel('Gradient norm');
192 legend('||g(x^{k})|| (Nesterov)', ...
193       '||g(x^{k})|| (Gradient, t=1/M)', ...
194       '||g(x^{k})|| (Gradient, t=2/(M+m))', ...
195       'FontSize', 14, 'Location', 'NorthEast');
196 grid on

```

D MATLAB code Nesterov's optimal gradient method

```
1 function [f_all, gnorm_all, x_all] = nesterov(fun, x0, M, q, tol, maxit)
2 % Code for Accelerated Gradient Method (Nesterov's Optimal Gradient Method)
3
4 % INPUT : 'fun' : anonymous function
5 %         'x0' : starting point
6 %         'q' : parameter
7 %         'tol' : tolerance threshold on gradient norm (stopping criteria)
8 %         'maxit' : max number of iterations
9
10 % OUPUT : 'f_all' : sequence of objective function values
11 %         'gnorm_all' : sequence of gradient norms
12
13 % To store values of x, f(x) and norm(g(x))
14 f_all = [];
15 gnorm_all = [];
16 x_all = {};
17
18 % Initialize current xk and yk at the starting point x0
19 xk = x0;
20 yk = x0;
21
22 % Main loop of the gradient descent
23 k=1;
24 while (k ≤ maxit)
25
26     % Evaluate the anonymous function & gradient at current yk
27     [f_yk, g_yk] = fun(yk);
28
29     % Record yk and its value
30     f_all = [f_all; f_yk];
31     gnorm_all = [gnorm_all; norm(g_yk)];
32     x_all{end+1} = yk;
33
34     % Stop if we are below tolerance level
35     if norm(g_yk) < tol
36         break
37     end
38
39     % Take the step for x and y
40     xkp1 = yk - (1/M)*g_yk;
41     ykp1 = xkp1 + q*(xkp1 - xk);
42     xk = xkp1;
43     yk = ykp1;
44     k = k+1;
45 end
```

E MATLAB code fixed step size gradient descent

```
1 function [f_all, gnorm_all, x_all] = gradmeth(fun, x0, t, tol, maxit)
2 % Code for Accelerated Gradient Method (Nesterov's Optimal Gradient Method)
```

```

3
4 % INPUT : 'fun' : anonymous function
5 %         'x0' : starting point
6 %         't' : fixed step size
7 %         'tol' : tolerance threshold on gradient norm (stopping criteria)
8 %         'maxit' : max number of iterations
9
10 % OUPUT : 'f_all' : sequence of objective function values
11 %         'gnorm_all' : sequence of gradient norms
12 %         'x_all' : sequence of points
13
14 % To store values of x, f(x) and norm(g(x))
15 f_all = [];
16 gnorm_all = [];
17 x_all = {};
18
19 % Initialize current xk at the starting point x0
20 xk = x0;
21
22 % Main loop of the gradient descent
23 k=1;
24 while (k ≤ maxit)
25
26     % Evaluate the anonymous function & gradient at current xk
27     [fk, gk] = fun(xk);
28
29     % Record xk and its value
30     f_all = [f_all; fk];
31     gnorm_all = [gnorm_all; norm(gk)];
32     x_all{end+1} = xk;
33
34     % Stop if we are below tolerance level
35     if gnorm_all(k) < tol
36         break
37     end
38
39     % Compute the descent direction at xk (= negative gradient)
40     dx = -gk;
41
42     % Take the step
43     xk = xk + t*dx;
44     k = k+1;
45 end

```