MAT3007 Assignment 7

A7.1

(a)

We have

$$\nabla f(x) = \begin{bmatrix} -400x_1(x_2 - x_1^2) + 2x_1 \\ 200(x_2 - x_1^2) \end{bmatrix},$$
$$\nabla^2 f(x) = \begin{bmatrix} -400(x_2 - 3x_1^2) + 2 & -400x_1 \\ -400x_1 & 200 \end{bmatrix}.$$

MATLAB code for (a), (b):

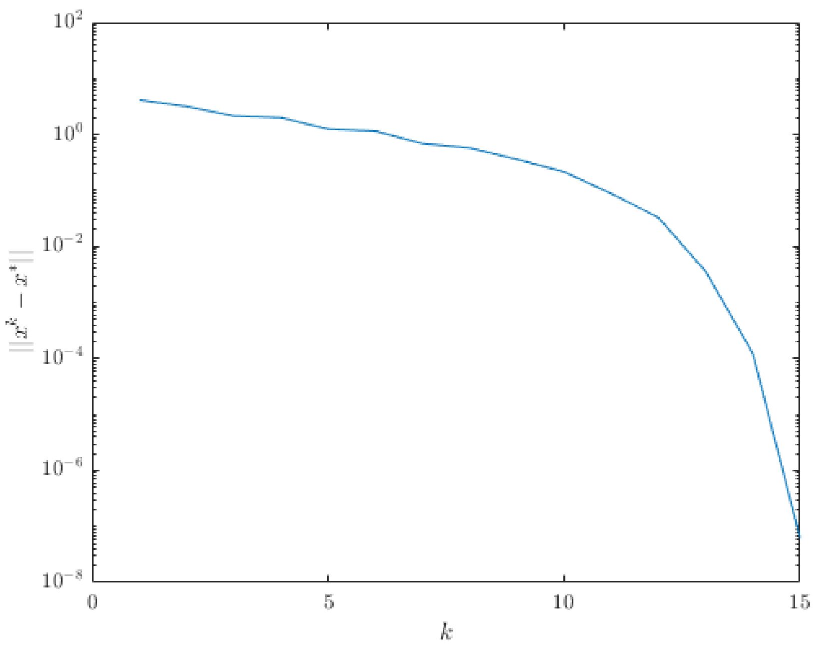
```
g   = @(x) x(2) - x(1)^2;
h   = @(x) 1 - x(1);

f   = @(x) 100 * g(x)^2 + h(x)^2;
df  = @(x) [-400 * x(1) * g(x) - 2 * h(x); 200 * g(x)];
hf  = @(x) [-400 * ( x(2)-3*x(1)^2 ) + 2, -400 * x(1);
-400 * x(1), 200];

[x, fx, iter, xs] = global_newton(f, df, hf, [2;5], 1e-6,
1/2, 0.1, 1e-6, 0.1);
fprintf("Global Newton: Minimum %f found at x = (%f,%f)
after %d iterations.\n", fx, x', iter);
diffs = xs - repmat(x', size(xs,1), 1);
dists = norms(diffs, 2, 2);
plot(dists);

function [x, fx, iter, xs] = global_newton(f, df, hf,
init, tol, sigma, gamma, g1, g2)

    % minimize function f using gradient the globalized
newton method.
    % args: f: function handle
    %       df: function gradient
    %       hf: function hessian
    %       init: initial point
    %       tol: stopping tolerance
    %       sigma, gamma: backtracking parameters
    %       g1, g2: parameters for descent condition

    iter    = 1;
    x       = init;
    fx      = f(x);         % function val at x
    dfx     = df(x);        % gradient val at x
    hfx     = hf(x);        % hessian val at x
    s       = -hfx\dfx;     % newton direction at x
    nrm     = norm(dfx);
    xs      = zeros(15, 2);
    xs(1,:) = x';

    while (nrm > tol)
        fprintf("iter:%02d      x:(%f,%f)     norm:%f
optval:%f", ...
                 iter, x(1), x(2), nrm, fx);

        % test descent condition
        if (-dfx' * s >= g1 * min([1, norm(s)^g2]) *
norm(s)^2)
            dir = s;
            fprintf("...Newton\n");
        else
            dir = -dfx;
            fprintf("...Gradient\n");
        end

        % backtrack to decide stepsize
        step = 1;
        while ( f(x+step*dir) - f(x) > -
gamma*step*dfx'*dir )
            step = step * sigma;
        end

        % update everything
        iter        = iter + 1;
        x           = x + step * dir;
        fx          = f(x);
        dfx         = df(x);
        hfx         = hf(x);
        s           = -hfx\dfx;
        nrm         = norm(dfx);
        xs(iter,:)  = x';

    end

    iter = iter - 1;
    xs = xs(1:iter,:);
```

```
end
```

```
iter:01       x:(2.000000,5.000000)     norm:822.680983
optval:101.000000...Newton
iter:02       x:(2.005025,4.020101)     norm:2.030309
optval:1.010076...Newton
iter:03       x:(1.755031,3.017619)     norm:47.087270
optval:0.960895...Newton
iter:04       x:(1.699116,2.883869)     norm:3.578211
optval:0.489741...Newton
iter:05       x:(1.484043,2.154565)     norm:30.873322
optval:0.462969...Newton
iter:06       x:(1.438223,2.066386)     norm:2.126150
optval:0.192480...Newton
iter:07       x:(1.283909,1.623558)     norm:14.233250
optval:0.142419...Newton
iter:08       x:(1.236373,1.526358)     norm:1.653217
optval:0.056383...Newton
iter:09       x:(1.154973,1.326207)     norm:4.190653
optval:0.030032...Newton
iter:10       x:(1.094227,1.193642)     norm:1.948752
optval:0.010240...Newton
iter:11       x:(1.040012,1.078686)     norm:1.429247
optval:0.002465...Newton
iter:12       x:(1.014813,1.029211)     norm:0.314192
optval:0.000260...Newton
iter:13       x:(1.001669,1.003168)     norm:0.080365
optval:0.000006...Newton
iter:14       x:(1.000056,1.000109)     norm:0.001265
optval:0.000000...Newton
iter:15       x:(1.000000,1.000000)     norm:0.000001
optval:0.000000...Newton
Global Newton: Minimum 0.000000 found at x =
(1.000000,1.000000) after 15 iterations.
```

We see that the Newton direction was always chosen.

(b)



A quadratic convergence.

(c)

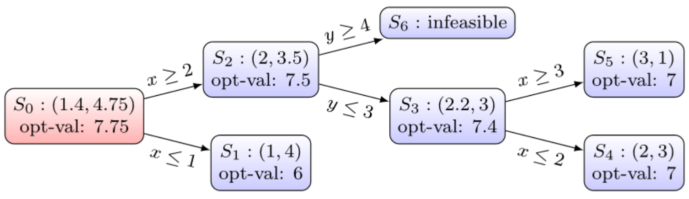
Using the code for Assignment 6, we obtain the output

```
Gradient Descent: Minimum 0.000000 found at x =
(1.000001,1.000002) after 20373 iterations.
```

which is significantly slower than the Newton method.

A7.2

At each node we add new constraints (specified on the edges) and branch into two disjoint relaxed LP. We then solve the two relaxed LP recursively until we've reached an integer solution.


$S_0 : (1.4, 4.75)$ opt-val: 7.75
$x \geq 2$
$x \leq 1$
$S_2 : (2, 3.5)$ opt-val: 7.5
$S_1 : (1, 4)$ opt-val: 6
$y \geq 4$
$y \leq 3$
S_6 : infeasible
$S_3 : (2.2, 3)$ opt-val: 7.4
$x \geq 3$
$x \leq 2$
$S_5 : (3, 1)$ opt-val: 7
$S_4 : (2, 3)$ opt-val: 7

We obtain the optimal solutions $(x, y) = (2, 3)$ and $(3, 1)$ with optimal value 7.

A7.3

(a)

Define $X := [x_{ij}]$ where x_{ij} denotes whether item i is in knapsack j. Also collect the v_i, a_i, C_j into vectors v, a, C, respectively. We wish to

$$\begin{aligned} \max_X \quad & v^\top X\mathbf{1} \\ \text{subject to} \quad & X\mathbf{1} \leq \mathbf{1} \\ & X^\top a \leq C \\ & x_{ij} \in \{0,1\} \qquad \forall i,j \end{aligned} \tag{1}$$

where $\mathbf{1}$ denotes all-one vectors of suitable sizes.

(b)

For the IP, we have $v = [2,1,3,2,1,4,2]^\top$, $a = [2, 0.5, 0.5, 0.1, 0.5, 1, 1.5]^\top$, and $C = [3,2]^\top$ in (1).

```
v = [2;1;3;2;1;4;2];
a = [2;0.5;0.5;0.1;0.5;1;1.5];
C = [3;2];
```

Solving,

```
cvx_begin
variable X(7,2) binary
maximize( v' * X * ones(2,1) )
subject to
    X  * ones(2,1) <= ones(7,1);
    X' * a <= C;
cvx_end
```

we obtain

$$X^* = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \text{ opt-val} = 13.$$

If we relax the binary constraint to $x_{ij} \in [0,1]$,

```
cvx_begin
variable X(7,2) nonnegative
maximize( v' * X * ones(2,1) )
subject to
    X  * ones(2,1) <= ones(7,1);
    X' * a <= C;
    X <= ones(7,2);
cvx_end
```

we instead obtain

$$X^* = \begin{bmatrix} 0.3319 & 0.1181 \\ 0.5350 & 0.4650 \\ 0.5317 & 0.4683 \\ 0.5062 & 0.4938 \\ 0.5350 & 0.4650 \\ 0.5656 & 0.4344 \\ 0.6127 & 0.3873 \end{bmatrix}, \text{ opt-val} = 13.9.$$

The integrality gap is 0.9.